Direct Policy Transfer via Hidden Parameter Markov Decision Processes

Jiayu Yao [*1] **Taylor Killian** [*1 2] **George Konidaris** [3] **Finale Doshi-Velez** [1]

1. Introduction

Many situations arise in which an agent must learn to solve tasks with similar, but not identical, dynamics. For example, a robot might be tasked to manipulate objects with slightly different masses and volumes; a clinician may treat patients with unique—but still all human adult—physiologies. In such situations, if one has already seen several instances of the related tasks, it is inefficient to start learning a new task from scratch. Indeed, in some domains, such as medicine, one does not have multiple episodes to learn a personalized treatment policy: decisions must be optimized from only a few interactions with the patient.

The recently introduced Hidden Parameter Markov Decision Process (HiP-MDP) (Doshi-Velez and Konidaris, 2016; Killian et al., 2017) addresses this setting by learning a low-dimensional representation w_b for the transition dynamics of each instance b, i.e, the transition function can be modelled as $T(s'|s, a, w_b, \mathcal{W})$, where $\mathcal{W}$ are some global parameters. However, learning models that are accurate enough for personalized planning is challenging. When encountering a new instance, Killian et al. (2017) required *a few full episodes* in the environment to learn the local embedding w_b and additionally refine $\mathcal{W}$, parameters of the underlying transition dynamics. Moreover, even after a sufficiently accurate model was learned, identifying a near-optimal policy was computationally expensive since it required training a Deep-Q Network (DQN) with a large number of simulated episodes of that specific task instance.

Our core insight is that perhaps the entire policy-learning process can be short-circuited if we use the embedding w_b as input into a policy $\pi(a|s, w_b)$. We hypothesize that after observing only a few transitions of a new instance b, the embedding w_b may be sufficient to *directly* parameterize a near-optimal policy

We demonstrate policy transfer in a toy 2D navigation domain, acrobot (Sutton and Barto, 1998), and with simulated treatment protocols for patients with HIV (Ernst et al., 2006). Our procedure results in immediate performance improvement after a small number of interactions with the environment of a new task instance, with further improvement as more transitions are observed, at a fraction of the computational cost.

2. Related Work

There exists a broad literature on transfer learning within reinforcement learning via latent representations (e.g. (Taylor and Stone, 2009; Hausman et al., 2018; Ha and Schmidhuber, 2018; Narasimhan et al., 2017; Morton and Kochenderfer, 2017)). Our work falls in the category of work where the goals do not change, but the dynamics of the system do. In this regime, there are problems that involve transfer across large changes in the state or action space (e.g. transferring policies across agents with different numbers of actuators (Gupta et al., 2017)). Hausman et al. (2018) formulate a separate embedding space where tasks with differing dynamics could be directly compared and composed in order to accelerate the learning of hierarchical or other complex tasks. Unlike these efforts, we focus on situations where there are subtle variations in the transition function, and we want to adapt to those variations quickly. Closest to our work, Killian et al. (2017) accounts for the variation in the transition dynamics directly by learning a general transition function with embedded latent parameters that represent variations in the dynamics. Zhang et al. (2018) decouples the dynamics and rewards when learning a task and is thus capable of transferring to problems where either the dynamics and rewards mechanism may change. However, both approaches are still model-based, in that they learn or retrain a model for a new instance and then must solve it to

[*]Equal contribution [1]School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, USA [2]MIT Lincoln Laboratory, Lexington, MA, USA [3]Department of Computer Science, Brown University, Providence, RI, USA. Correspondence to: Jiayu Yao <jiy328@g.harvard.edu>, Taylor Killian <taylorkillian@g.harvard.edu; taylor.killian@ll.mit.edu>.

DISTRIBUTION STATEMENT A.
This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

The 2^{nd} Lifelong Learning: A Reinforcement Learning Approach (LLARLA) Workshop, Stockholm, Sweden, FAIM 2018.

determine a policy. By contrast, our direct policy transfer approach avoids solving a model at test-time, and does not require the estimated dynamics to be accurate to identify a well-performing policy.

3. Background

Model-based reinforcement learning & HiP-MDPs Hidden parameter MDPs, or HIP-MDPs (Doshi-Velez and Konidaris, 2016; Killian et al., 2017), describe a *family* of Markov Decision Processes (MDPs) via the tuple $\{S, A, W, T, R, \gamma, P_W\}$, where $S \subseteq \mathbb{R}^D$ is the set of continuous states s, A is the set of discrete actions a, and R is the reward function. For each instance b in the family parameterized by the hidden parameters $w_b \sim P_W$, it acts in a continuous state space $S \subseteq \mathbb{R}^D$ using a discrete action space A. Assume the true transition dynamics of each instance, $T(s'|s, a, w_b)$, is unknown and that we are given a reward function $R(s, a) : S \times A \to \mathbb{R}$. The HiP-MDP framework assumes that a finite-dimensional array of hidden parameters w_b can fully specify variations of the true task dynamics. It also assumes the system dynamics do not change during a task and the agent is aware of when one task ends and another begins. In model-based reinforcement learning, the goal is to estimate the transition function $\hat{T}(s'|s, a, w_b)$ from observed transitions (s, a, s') of the new instance and then use the model $\hat{T}(s'|s, a, w_b)$ to learn a policy $a = \pi_b(s)$ that maximizes long-term expected rewards $E[\sum_t \gamma^t r_t]$, where $\gamma \in (0, 1]$ governs the relative importance of immediate and future rewards.

4. Direct Policy Transfer via HiP-MDPs

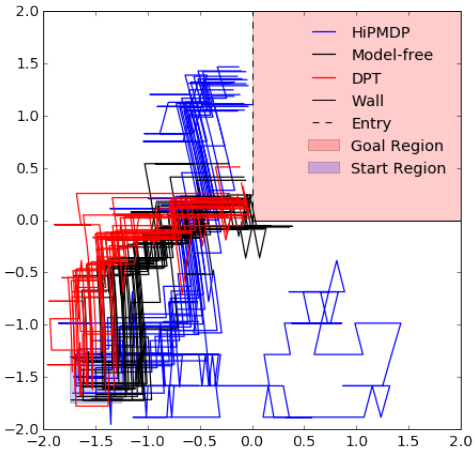
The HiP-MDP assumes that the transition dynamics can be modeled via a function $T(s'|s, a, w_b)$, where w_b is an instance-specific latent parameter. Placing a Gaussian prior on w_b gives us the following generative model:

$$
\begin{aligned}
s' &\sim T(s, a, w_b, \mathcal{W}) + \epsilon \\
w_b &\sim \mathcal{N}(\mu_w, \Sigma_b) \\
\epsilon &\sim \mathcal{N}(0, \sigma_n^2)
\end{aligned}
$$

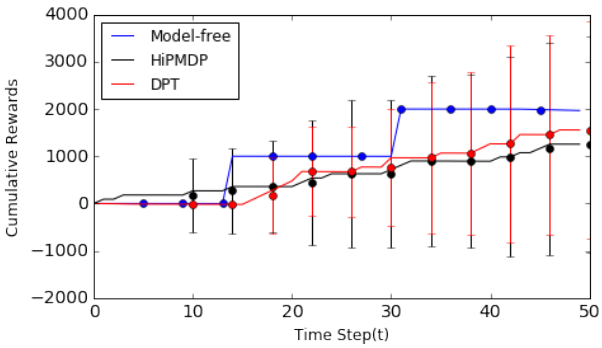
where $\mathcal{W}$ are the parameters of the transition model. We model the transition function T with a BNN and maintain a posterior $p(\mathcal{W})$ over the global transition parameters $\mathcal{W}$.

When starting a new instance b, Killian et al. (2017) updated estimates of w_b and $\mathcal{W}$ based on interactions with the environment. Next, trajectories are simulated via the BNN approximation of the transition function $\hat{T}(s, a, w_b|\mathcal{W})$ to train a Double Deep Q Network (DDQN) (Hasselt et al., 2016). Using the approximate transition model for planning was efficient in the sense that it limited the data needed from the real environment. However, training the DDQN required 500-1000 simulation episodes for each new instance, making the approach computationally inefficient. Additionally, if the accuracy of $\hat{T}$ was poor, the simulated episodes diverged from the dynamics of the environment, and the resulting policies were sub-optimal. Killian et al. (2017) required several episodes of real experience with a new instance b to obtain transition models that were accurate enough for planning; ideally we would be able to start behaving near-optimally within the first episode.

Rather than use the approximate transition model $\hat{T}$ to plan, we propose to use the hidden parameters w_b for direct policy transfer. The core hypothesis is that the hidden parameters w_b, while perhaps insufficient for model-based planning, still contain information about the dynamics of the environment and the similarities between previous instances. This information may then be used to key the policy to directly account for changes in the underlying dynamics of the task.



(a) A comparison of epsilon greedy policies π_{DDQN}, π_{HiPMDP}, π_{DPT} ($\epsilon = 0.15$)



(b) A comparison of cumulative rewards of multiple runs following the three policies. We extract π_{DDQN}, π_{HiPMDP}, π_{DPT} ($\epsilon = 0.15$) after training and ran each policy for 50 steps. We allow the agent to restart if reaching the terminal state for easy comparision.

Figure 1. A demonstration of the efficiency benefits of direct policy transfer using a simple 2D navigation domain

Specifically, we will learn a policy function

$$a = \pi(s, w_b; \mathcal{Y})$$

where $\mathcal{Y}$ are the function's global parameters. Given π, as soon as we have collected sufficient experience from the new task instance to identify w_b, we expect the policy to perform near-optimally.

To demonstrate the utility of embedding the parameters w_b with the input to the policy π, we compare the proposed method with the HiP-MDP (Killian et al., 2017) and model-free (DDQN) approaches in a simple 2D navigation domain, introduced by (Killian et al., 2017), in Figure 1.

The model-free policy π_{DDQN} (learned with an ϵ-greedy policy with $\epsilon = 0.15$ and 500 episodes within the real environment) learns a better route than the HiP-MDP policy π_{HiPMDP} obtained from the BNN-based transition function trained with 1000 observed state transitions of the real environment. The policy from our proposed direct-transfer approach π_{DPT} is on par with (perhaps even slightly better than) the model-free DDQN policy: The average cumulative rewards of π_{DDQN}, π_{HiPMDP}, π_{DPT} are 994.8, 943.6, 992.7 respectively. On average, π_{HiPMDP} takes more steps to reach the goal than the other approaches (Figure 1(a)), and thus accumulates less rewards while π_{DPT} is near-optimal.

5. Inference

We assume that we are provided a large batch of data from previously observed instances, including near-optimal policies for each instance. This is reasonable in many domains: we may have experimented with swinging many different kinds of robotic arms and found good swing-up policies for them; we may have treated large numbers of patients and made them well. The objective is to use this information to rapidly perform near-optimally on any new instance. Specifically, in the following, we assume that estimates of the global parameters, $p(\mathcal{W})$ and $\mathcal{Y}$, are learned from this initial batch of data and then kept fixed; at test-time, one may only determine the local hidden parameters w_b. These assumptions match industrial settings where one may wish to deploy a system in a variety of locations and circumstances and not want unexpected global changes post deployment (e.g. (Depeweg et al., 2016)). That is, we only want our customization to alter the control procedure for the individual circumstances each deployed unit operates in.

We denote the set of N previously observed instances as $\{1, 2, ..., n, ..., N\}$ and the new instance as b. Each instance n provides a collection of observed transitions $\mathcal{D}_n = \{(s^n, a^n, r^n, s'^n)\}$; and we denote all the observed transitions in the initial batch by $\mathcal{D} = \{(s^n, a^n, r^n, s'^n)\}_{n=1}^N$. We denote the near-optimal polices for each instance $n \in N$ as $\pi_n^*(s)$ and the collection of all the policies as Π^*. (In our experiments, each π_n^* is obtained via online learning; these could also be found by observing experts or any other procedure.)

5.1. Batch Training of Global Parameters

Learn $\hat{T}$, the BNN-based transition function, the w_n for each instance and $p(\mathcal{W})$. When given a batch of transitions $\{(s^n, a^n, r^n, s'^n)\}$, we have all the elements needed to learn the BNN-based transition function $\hat{T}$. We first create a set of inputs to the BNN $\{(s^n, a^n, w_n)\}$ and a set of outputs $\{s'^n\}$, where the w_n are hidden variables with the same value for all transitions from instance n. Next, we use the expectation-propagation based approach of Hernández-Lobato et al. (2016) to obtain a fully factorized posterior $p(\mathcal{W}|s^n, a^n, s'^n, w_n) \approx \prod_i q(\omega_i) \equiv p(\mathcal{W})$ where the ω_i's are the weights in the network and maximum-likelihood local hidden parameters $\{w_n\}$ (in the optimization, we iterate between updating the posterior on $\mathcal{W}$ and $\{w_n\}$). We keep a posterior on the global parameters $\mathcal{W}$ because this uncertainty in $\mathcal{W}$ encapsulates the stochasticity in the environment.[1] In initial experiments, we found that maximum-likelihood estimates of the local parameters were sufficient because their posteriors were typically sharply peaked, even after just a few iterations of experience.

In some situations (such as in the experiments presented in Section 7), we may have choices in how the initial batch of data is collected. In our case, we run an independent DDQN for each instance n (with ϵ-greedy exploration, $\epsilon = 0.15$), updating the policy as we go; this allows us to collect transitions in regions that are most relevant for performing well on the task.

Learn the parameters $\mathcal{Y}$ of the policy $\pi(s, w_n; \mathcal{Y})$. In some settings, we may be given a set of near-optimal policies Π^*. In others, these may be learned during some initial training phase. For example, once we have a trained DDQN (as described above) for each instance, we can exploit its optimal policy by setting $\epsilon = 0$. The next step is to learn the policy $\pi(s, w; \mathcal{Y})$. Starting with the observed $\{(s^n, w_n)\}$ in each batch, we train a multi-layer perceptron (MLP) to predict the action $a^* = \pi_n(s_n)$, effectively distilling the model-free policy (Rusu et al., 2015), keyed by the learned w_n. It is expected that if $\pi(s, w_n; \mathcal{Y})$ has been trained with a sufficient number of observed instances n, then for any new instance b, $\pi(s, w_b; \mathcal{Y}) \approx \pi_b^*(s)$.

5.2. Direct Policy Transfer at Test Time

When encountering a new instance, a new hidden parameter setting is initialized from the mean of batch instances $w_b = \mathbb{E}[w_n], n \in N$. Initially the policy acts in an ϵ-greedy

[1] An alternative would be to use input-uncertainty BNNs, as done by Depeweg et al. (2016).

fashion with respect to $\pi(s, w_b; \mathcal{Y})$. The ϵ-greedy exploration allows to better capture information about different dynamics and thus accurately estimate w_b. As we gather more experience with the environment, we periodically update the hidden parameters w_b and reduce exploration. As our estimates of w_b converge, our policy $\pi(s, w_b; \mathcal{Y})$ adapts to the new instance of the task.

To update w_b, we maintain a buffer of all transitions observed in the current instance b: $\{(s^b, a^b, s'^b)\}$. With these observations and the posterior over BNN weights $p(\mathcal{W})$, we optimize the latent parameters w_b to minimize the α-divergence of the approximate posterior distribution $p(\mathcal{W})$ and $p(\mathcal{W}|s, a, w_b, s')$. In Section 7, we see that this process enables direct transfer, providing near-optimal decisions with only a few interactions in the environment.

6. Experimental Setup

Evaluation and Baselines We consider two evaluation measures: computational time and cumulative rewards over a fixed number of iterations. (We emphasize that the number of iterations are generally on the order of an episode, much smaller than in earlier work.)

We compare against two baselines that also use latent parameters, one that is model-based and one that is policy-based. We do not compare to average model or online model-free learning with just the environment because Killian et al. (2017) demonstrated that the HiP-MDP dominated those approaches in their work. The model-based HiP-MDP baseline (labeled in the figures as simply HiP-MDP), upon updating its hidden parameters, performs rollouts from the approximate transition function to update a DDQN policy. The second is a PCA-based approach to creating a latent representation for direct policy transfer. All approaches follow the same latent parameter update schedule.

The PCA-baseline is motivated by the fact that perhaps there is a simpler way to capture statistics about the environment that are sufficient for planning, rather than via the latent parameters of the transition function. That is, is there a simpler dimensionality reduction that also works? As a baseline, we define the transition statistics of an instance ψ_n as the average change in the observed state after performing each action (Δs_{a_i}). That is: $\psi_n = [\Delta s_{a_1}, \ldots, \Delta s_{a_M}]$. Then, with N previous instances, we form the matrix Ψ_N, where each row corresponds to the transition statistics, ψ_n of a separate instance. Then,

$$\Psi_N = U_\Psi S_\Psi V_\Psi^\intercal$$

where V_Ψ represents the principal directions of the new basis that can be used to project the collected transition statistics from a new instance ψ_b into a low dimensional form w_b^{PCA}. That is,

$$w_b^{PCA} = \psi_b \cdot V_\Psi$$

Once we have w_b^{PCA}, we train a policy $\pi^{\text{PCA}}(s, w_b^{PCA}; \mathcal{Y}^{PCA})$ analogously as we did for our approach.

Domains We evaluate our direct transfer approach in three domains: a 2D Navigation task, Acrobot, and simulated HIV treatment, the same domains used by Killian et al. (2017) chosen to directly compare with the HiP-MDP.

2D Navigation. We revisit the 2D navigation domain from the demonstration in Figure 1. The training batch was collected from two instances, one for each class ($\theta_b = 0,\ \theta_b = 1$). We set the dimension of the latent weights w_n to three. We tested the algorithms with two instances, one for each class. For HiP-MDP in the 2D navigation domain, as mentioned in Section 4, we observed that the DDQN often got stuck in local optima. In initial experiments, we also observed with small ϵ, if the agent starts close to the wall, it continually bumps into the wall, collecting negative rewards, and is unable to gather enough information to learn an accurate parameter representation w_b. Thus, we set the $\epsilon_{\text{min}} = 0.15$ during the test time. The latent parameters, w_b, were updated after every 5 transitions.

Acrobot. It has been shown that in the acrobot domain, a policy will generalize poorly if subtle changes are made to the dynamics (Bai et al., 2013), and thus this domain is well suited for investigation for transfer among families of tasks. We set the latent weight dimension to 5 based on preliminary experiments. Our training batch consists of 8 previous instances. During test time, ϵ is set to zero for all approaches. We collect 20 transitions before updating w_b to ensure there is sufficient information to estimate the dynamics.

HIV treatment. In this domain, the task is to determine an effective HIV treatment for patients with different physiological dynamics; the dynamics involve 22 latent parameters and thus are significantly more complex than in the previous domains. We set the dimension of the latent parameters, w_b, as 5. We train on a batch of 10 patients and test on 2 new patients. It is expected that a more complex domain more information to obtain a fairly accurate w_b. However, as we tracked some patients' physiological state following a near-optimal policy, we discovered out that the behavior follows a periodic pattern. The average period is $20 - 40$ time steps. Hence, in the testing time, we collected 3 transitions over 10 training epochs to estimate w_b. In testing, $\epsilon = 0.0$.

Additionally, the individual instances of the HIV simulator vary widely in terms of the state and rewards distributions. Rather than aggregate average performance across all test instances, we separate the two test instances to demonstrate the performance of the proposed evaluation approaches on each one individually.

Training Procedure For each domain, we collected a batch of data $\mathcal{D} = \{s^n, a^n, r^n, s'^n\}_{n=1}^N$ from instances

Algorithm 1 Direct Policy Transfer (DPT) through a HiP-MDP framework

Training Phase:
Input: Observed Transitions $\mathcal{D}$, a set of near-optimal policies Π^*
function TuneModel($\mathcal{D}, \mathcal{W}, \{w_n\}$)
  for Number of Epochs **do**
    Update $\mathcal{W}$ given $\mathcal{D}$
    Update each w_n given $\mathcal{D}_n$
  end for
  return $p(\mathcal{W})$,$\{w_n\}$
end function
function LearnJointPolicy($\Pi^*, \{w_n\}$)
  for Number of Epochs **do**
    Update $\pi(s, w_n, \mathcal{Y}) \rightarrow \pi_n(s)$
  end for
  return $\mathcal{Y}$
end function
Testing Phase:
Input: Observations $\mathcal{D}_b$, $p(\mathcal{W})$, $\mathcal{Y}$,$\{w_n\}$
main Direct-Policy-Transfer
  Init. $w_b = E[w_n]$
  for $i = 0$ to t steps **do**
    Take action $a \leftarrow \pi(s, w_b, \mathcal{Y})$
    Store $\mathcal{D}_b \leftarrow (s, a, r, s', w_b)$
    if Time to Update w_b **then**
      Update w_b given $\mathcal{D}_b$, $p(\mathcal{W})$
    end if
  end for
end main

with various dynamics using the DDQN for 500 iterations with $\epsilon_{\text{min}} = 0.15$, decaying epsilon after each episode, with a learning rate of 5×10^{-4}. Using $\mathcal{D}$, we update $\mathcal{W}$ and w_b iteratively for 60 epochs. The learning rate of BNN in the 2D navigation domain is 5×10^{-4} and is 2.5×10^{-4} for all other domains. We extracted the DDQN policy from the previous step, set ϵ to zero, and generate trajectories of 50 episodes within each instance. With these data samples, we train a MLP with one hidden layer of 32 nodes to generate a joint policy for all instances. During testing time, at each iteration, we collect only a few times steps into the buffer and then update the latent weights, which allows us to adjust the policy. We updated the embedding parameters over 10 iterations, in total.

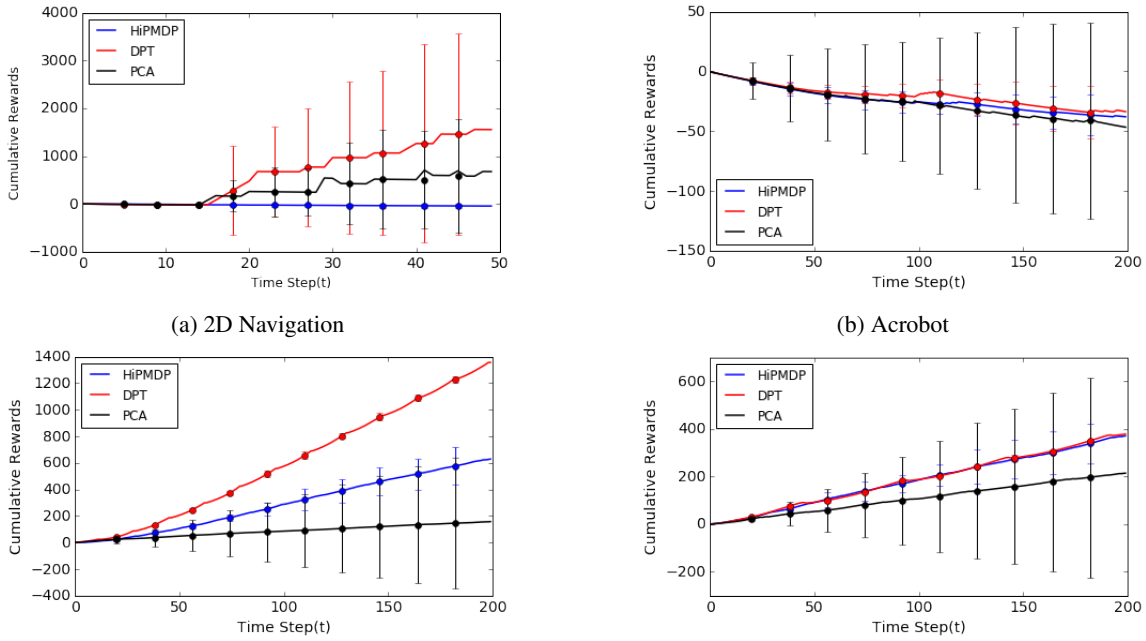
7. Results

As discussed in Section 6, we sought to improve upon the HiP-MDP in both performance and computational time by leveraging the polices developed for previously observed instances and transferring them directly to the new instance. In Table 1 we report the computation time and performance achieved when an agent first encounters a new instance of a task (averaged over 5 separate runs). Computation time measures the time the agent spent operating in the first episode of the new instance while developing an optimal control policy, including inference of the latent parameters w_b and then computing the control policy.

PCA does not sufficiently capture the dynamics of newly encountered variations. Unsurprisingly, the PCA baseline is the fastest computationally, as the inference of the parameter w_b^{PCA} only requires a single matrix computation to project the observed transition statistics to the pre-computed basis. However, the performance of the PCA baseline is significantly worse than the other approaches. It is unclear whether the features preserved via PCA contain sufficient information to differentiate between variations in the dynamics of the environment, and the estimates were highly dependent on the specific state transitions observed (that is, the estimate of w_b^{PCA} could vary widely between runs). This reinforces the utility of training a model of the transition dynamics: our approach, which infers the latent w_b via the BNN, had parameters designed to be informative about the dynamics of the current instance.

The direct policy transfer approach requires fewer interactions with the environment to learn a high-performing policy. By using the latent estimate w_b as input to the policy derived through DPT, we were able to identify a near-optimal policy with less data than when using the HiP-MDP alone. This is demonstrated in Figure 2 as DPT, the red curve, which adapts to the new task instance sooner thus achieving better performance in the initial episode of the new instance. This indicates that DPT adapts to the change in transition dynamics more rapidly than either the HiP-MDP or PCA baseline. Average cumulative reward over test episodes is reported in the Table 1. While the variances are sometimes high, the recorded performance demonstrates the advantage DPT gains by learning the specific variation present in the new instance earlier than in the HiP-MDP. (Or, as in the second HIV test instance, achieving comparable performance.)

At test time, the direct policy transfer approach is faster than the original HiP-MDP. As recorded in the Table 1, DPT provides an overwhelming improvement in computational efficiency over the HiP-MDP, representing 8-16x reduction in computational time, needing only approximately 20 minutes to replace several hours of computation. The primary contributor toward this improvement comes through avoiding the need to perform extensive rollouts of the ap-



(a) 2D Navigation

(b) Acrobot

(c) HIV Test Patient 1

(d) HIV Test Patient 2

Figure 2. Cumulative rewards achieved throughout the initial episode of a newly encountered instance. The early improvement of the red curve indicates that DPT adapts to the change in transition dynamics more rapidly than either the HiP-MDP or PCA baseline.

Table 1. Experimental results where DPT is evaluated against the HiP-MDP and PCA baseline

	Computation Time			Cumulative Rewards			
	2D Nav	Acrobot	HIV	2D Nav	Acrobot	HIV	
PCA	**17.4s±0.52**	**56.3s±1.49**	**180.6s±4.43**	317.9±207.8	-42.7±38.89	100.8±12.8	207.8±1.53
HiPMDP	1.0×10^4s	1.9×10^4s	1.0×10^4s	809.9±35	-30.8±33.2	726.7±59.8	**580.0±21.9**
DPT	1.1×10^3s	1.2×10^3s	1.2×10^3s	**891.9±319**	**-27.7±49.5**	**1425.0±5.6**	562.2±4.2

proximated transition function to enable policy learning with the DDQN, as described above. This result reinforces the decision to leverage the previous set of policies, Π^*, to develop $\pi(s, w_n; \mathcal{Y})$ which extends to $\pi(s, w_b; \mathcal{Y}) \rightarrow \pi_b^*(s)$. (The PCA-based baseline is the fastest, but as we mention above, it produces policies that are significantly worse than either the HiP-MDP or the DPT approaches.)

8. Discussion and Conclusion

We used a latent variable model to capture differences in the dynamics of an environment, and then, rather than using that dynamical model for planning, used those parameters to directly parameterize a policy. This approach allowed us to learn statistics about the environment that captured the variation in dynamics across instances—essential for performing well—but was also both sample and computationally-efficient at test time: only a few interactions were needed to learn the parameters, and once learned, no expensive planning was required to perform well.

As with all latent variable-based approaches, an important questions is whether the latent variable—in this case the HiP-MDP hidden parameter—is sufficient for planning. Across our experiments, we found that this was the case, and the HiP-MDP parameter captured the necessary information about the environment more accurately than the PCA-based baseline. That said, it was essential to have sufficient variation among in training batches such that different enough parameter settings were encountered to generalize to those we may see in test. In situations where a large previously-collected cohort is available, this is not a large limitation. In some domains, it was also important to see a variety of actions (our use of the ϵ-greedy in the 2D navigation task, even at test time); it would be interesting to develop exploration policies to quickly and consistently identify the hidden parameter w_b in all environments, or heuristics for determining in what kinds of environments this kind of additional exploration is needed (and in what environments just taking any few actions is sufficient). Finally, currently our direct policy transfer approach does not adjust its policy

based on the rewards it receives during the new instance. It would be interesting to start with $\pi(s, w_b; \mathcal{Y})$ and then adapt that policy as we gather more data in the environment.

More broadly, our work makes important steps towards learning approaches to transfer policies from a batch of data, when future instances may not be exactly like the batch (and indeed, there are variations in the batch). We see interesting directions in considering end-to-end training that creates a transition-type model that is particularly well-suited to producing policy parameters, as well as imitation learning-based ways of identifying the original set of optimal policies Π^* when a simulator is not available. For safety-critical applications, such as healthcare, having even the rough transition model, in addition to the policy, may also provide a way to safe-guard against truly poor actions if regions where the policy is less accurate.

Acknowledgments

FDV, GDK, and JY acknowledge support from NSF RI-1718306.

References

H Bai, D Hsu, and WS Lee. Planning how to learn. In *Robotics and Automation, 2013 IEEE International Conference on*, pages 2853–2859. IEEE, 2013.

S Depeweg, JM Hernández-Lobato, F Doshi-Velez, and S Udluft. Learning and policy search in stochastic dynamical systems with Bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.

F Doshi-Velez and G Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, volume 25, pages 1432–1440, 2016.

D Ernst, G Stan, J Goncalves, and L Wehenkel. Clinical data based optimal STI strategies for HIV; a reinforcement learning approach. In *Proceedings of the 45th IEEE Conference on Decision and Control*, 2006.

A Gupta, C Devin, Y Liu, P Abbeel, and S Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. In *International Conference on Learning Representations*, 2017.

D Ha and J Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018. URL https://worldmodels.github.io.

H van Hasselt, A Guez, and D Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100. AAAI Press, 2016.

K Hausman, JT Springenberg, Z Wang, N Heess, and M Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

JM Hernández-Lobato, Y Li, M Rowland, D Hernández-Lobato, T Bui, and RE Turner. Black-box α-divergence minimization. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

TW Killian, SJ Daulton, G Konidaris, and F Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Neural Information Processing Systems*, 2017.

J Morton and MJ Kochenderfer. Simultaneous policy learning and latent state inference for imitating driver behavior. *arXiv preprint arXiv:1704.05566*, 2017.

K Narasimhan, R Barzilay, and T Jaakkola. Deep transfer in reinforcement learning by language grounding. *arXiv preprint arXiv:1708.00133*, 2017.

AA Rusu, SG Colmenarejo, C Gulcehre, G Desjardins, J Kirkpatrick, R Pascanu, V Mnih, K Kavukcuoglu, and R Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

R Sutton and A Barto. *Reinforcement learning: An introduction*, volume 1. MIT Press, Cambridge, 1998.

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

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