Transfer Learning by Discovering Latent Task Parametrizations

Finale Doshi-Velez
Harvard
Boston, MA 02115
finale@alum.mit.edu

George Konidaris
MIT CSAIL
Cambridge, MA 02139
gdk@csail.mit.edu

Abstract

We present a framework that is able to discover the latent factors that parametrize a family of related tasks from data. The resulting model is able to rapidly identify the dynamics of a new task instance, allowing an agent to flexibly adapt to task variations.

1 Introduction

In many real-world applications of learning for control, an agent is expected to repeatedly encounter tasks with dynamics that are similar, but never quite the same. For example, when learning to manipulate glasses of water, an agent might encounter glasses with different masses and different amounts of liquid. Similarly, when learning to drive a vehicle, an agent will encounter many different individual vehicles, each with (for example) brakes that behave slightly differently. Over the course of learning to fly a plane, an agent will encounter may different planes, each with slightly different handling characteristics, and carrying different loads.

In all of these scenarios, it makes little sense of the agent to start learning afresh when it encounters a new glass, a new vehicle, or a new plane. Indeed, one would hope that the more manipulation, driving, or flying tasks the agent performed, the more quickly and reliably the agent could adapt to new instances of the same type of tasks.

Tasks with these closely-related dynamics provide an interesting regime for transfer learning. Intuitively, these tasks seem to have some low-dimensional latent factors that parametrize the dynamics in structured ways: for example, the amount of liquid in the cup can be thought of a one-dimensional latent parameter which causes smooth changes in the cup's dynamics. Furthermore, in many interesting problems, either these latent parameters remain fixed for the duration of the task (the state of a car's brakes will not change significantly during a single trip), or the agent will know when a change has occurred (the agent is driving a different car).

Given knowledge of the latent parametrization—in the form of a generative model of the task dynamics given the latent parameters—an agent could rapidly identify the dynamics of a particular task instance by updating its belief over the latent parameters. Such an agent need not learn a new control policy for each task instance; it could instead either synthesize a parametrized control policy [7, 3] based on a point estimate of the latent parameter values, or learn or plan in the belief space over its parameters [9, 1, 11, 10, 16, 6].

This raises the question of how such latent parametrizations can be discovered from experience. We present preliminary results for a framework that leverages the knowledge of when the latent parameters may change to discover the number of latent parameters, and how they affect the task dynamics. We show that our framework can both infer the latent parameterization for a version of the cart pole problem where the length of the pole and its mass can vary between episodes, and rapidly identify the dynamics of a new task instance.

2 Approach

We assume that the data consists of input-output pairs (x_n^b, y_n^b) where x is a (possibly multi-dimensional) real vector, y is a real scalar, $b \in \{1, ..., B\}$ is the *batch* from which the data came, and $n \in \{1, ..., N_b\}$ is the index of the data within the batch. For example, the input x_n^b might be the agent's current position, velocity, and a force, and the output y_n^b might be the change in the agent's current position and velocity. Each batch b might represent a different mass that the agent had to carry.

We propose that the dynamics equation $y_n^b = g_b(x_n^b, w_b)$ for any task b can be parametrized by a small set of latent factors w_b. In special cases, we may be able to derive analytic expressions for how w_b affects the dynamics formula $g_b(x_n^b, w_b)$: for example, in the manipulation task, we might be able to derive the kinematic equations of how the cup will respond to a force given a certain volume of liquid. However, in most situations, the simplifications required to derive these analytical forms for the dynamics will be brittle at best.

We instead propose to learn the latent factorization from the data itself. Specifically, we fit a model of the form:

$$\begin{aligned} y_n^b &\sim \sum_k^K w_{kb} f_k(x_n^b, \theta) + \epsilon_n^b \\ \epsilon_n^b &\sim N(0, \sigma_n^2), \end{aligned}$$

where we can think of the w_{kb} as the values associated with the K latent factors and the f_k as the accompanying task-specific basis functions that describe how a change in the latent factor w_{kb} affects the overall dynamics. The additivity assumption in our semi-parametric basis function regression allows us to learn all the latent elements of this model: the number of factors K, the weights w_{kb}, and the form of the basis functions f_k.

Model We model this data the sum of functions f_k, $k \in 1...K$

$$\begin{aligned} y_n^b &\sim \sum_k^K w_{kb} f_k(x_n^b, \theta) + \epsilon_n^b \\ \epsilon_n^b &\sim N(0, \sigma_n^2), \end{aligned}$$

where w_{kb} is a real scalar corresponding to the weight of function f_k for batch b. We set $w_{1b} = 1$ for all batches b. We assume that the noise ϵ_n^b comes from the same distribution for all batches, and that the parameters θ are also the same for all functions.

We consider the following priors on the weights w_{kb} and the functions f_k:

$$\begin{aligned} f_k &\sim GP(\theta) \\ w_{kb} &\sim N(0, \sigma_w^2) \text{ for } k > 1 \\ w_{1b} &= 1, \end{aligned}$$

where GP is a Gaussian process. Here, θ are the parameters of the GP kernel (most importantly the length-scale) and α is the concentration parameter. We assume that the total number of functions K is bounded but unknown. The combination of a parametric (additive) structure with nonparametric basis functions results in the semi-parametric latent factor model, previously used by Teh, Seeger and Jordan [15] to learn correlated output processes.

The likelihood for this prior can be expressed in a fairly straight-forward manner. Let $N = \sum_b N_b$ be the total number of observations. We concatenate all the y_n^b into an N by 1 vector Y, and let $K(X, X)$ be the $N \times N$ matrix of all $K(x_n^b, x_{n'}^{b'})$. Let W be the $K \times B$ matrix of w_{kb} elements. Finally, we introduce an $B \times N$ indicator matrix A such that $A(b, n) = 1$ if the nth data-point came from batch b and zero otherwise. The marginal likelihood of the model is then:

$$Pr(Y|W) = N(0, K(X, X) \odot (A^T W^T W A) + I\sigma_n^2), \quad (1)$$

where $\odot$ is an element-wise or Hadamard product. The predictions $\hat{Y}$ are:

$$\hat{Y} = K(X, X) \odot (A^T W^T W A)(K(X, X) \odot (A^T W^T W A) + I\sigma_n^2)^{-1} Y.$$

Inference The only variable in the likelihood in equation 1 are the values of w_{kb}. We learn these through a straight-forward Metropolis-Hastings scheme with the proposal $w'_{kb} = w_{kb} + \delta_{kb}$, where $\delta_{kb} \sim N(0, \sigma^2_{MH})$. Since the proposal is symmetric, the acceptance threshold is simply

$$a = \min(1, \frac{P(Y|W')P(w'_{kb}|W_{-kb})}{P(Y|W)P(w_{kb}|W_{-kb})}),$$

where $P(Y|W)$ is given by equation 1, and $P(w_{kb}|W_{-kb}) = N(w'_{kb}; 0, \sigma^2_w)$.

In this work, we consider a scenario in which the agent is given a large amount of *batch* data from several different operational settings. These batch data are used to fit the weights W corresponding to the latent factors. Then the agent encounters a few data from a new scenario. We refer to these few data as the *training* data for that scenario. Given training data for a new batch b', we can learn the weights $w_{kb'}$ using the same MH procedure above, but holding the remaining weights fixed.

3 Results on Cartpole

We present initial results for cart pole, a standard reinforcement-learning task [12]. The cartpole task begins with a pole that is initially standing vertically on top of a cart; the agent's goal is to keep the pole from falling over by moving the cart. The agent has two available actions: apply a force either to the left or to the right of the cart. The full state-space of the cartpole domain is the position z and velocity $\dot{z}$ of the cart, as well as the angle θ and the angular velocity $\dot{\theta}$ of the pole. At each time step, the system evolves according to the following equations:

$$\begin{aligned} z_{t+1} &= \quad z_t + \tau \dot{z}_t \\ \dot{z}_{t+1} &= \quad v - ml\ddot{\theta}\cos\theta / M \\ \theta_{t+1} &= \quad \theta_t + \tau\dot{\theta}_t \\ \dot{\theta}_{t+1} &= \quad \dot{theta}_t + \tau\ddot{\theta}, \end{aligned}$$

where $v = \frac{f + ml\dot{\theta}_t^2 \sin\theta}{M}$, $\ddot{\theta} = (g \sin\theta - v\cos\theta)/(l(\frac{4}{3} - m\cos\theta^2/M))$, f is the applied force, g is gravity, M is the mass of both the cart and the pole, and m and l are the mass and length of the pole, respectively. We considered scenarios in which we varied m and l.

Varying the Pole Mass We collected 750 sample transitions from the cartpole task with the pole length held fixed to 0.5 and pole masses of 0.1, 0.2, and 0.3. These experiences were gathered by an agent learning to solve a specific task instance using Sarsa(λ) [12], and then subsampled for coverage. Each iteration of experience provided us with a four-dimensional input $x^b_n = \{z, \dot{z}, \theta, \dot{\theta}\}$ and a four-dimensional output $y^b_n = \{\Delta z, \Delta\dot{z}, \Delta\theta, \Delta\dot{\theta}\}$ describing the change in those inputs at the next time-step. We next used our approach to learn a latent parametrization and a set of Gaussian process basis functions for each output.

Next, we collected 50 sample transitions for cart masses of 0.1 , 0.15, 0.2, 0.25, and 0.3. These data represented a few trials on a new task where the agent did not know the mass or length of the pole. We used this new data to compute the weights associated with the new task for each action. The quality of the fit was assessed using 50 additional test points from the same task; we compared our approach to trying to generalize only from the 50 input points and also generalizing directly from the 2250 batch data points as a single Gaussian process (that is, ignoring the structure in the batch data).

Figure 1 shows a comparison of mean-squared error (MSE) for all four outputs for a single action (left). We see that in all cases, learning the latent parametrization—which leverages both the small amount of training data for each problem as well as the structure in the larger batch data, performs better than using the training data alone or ignoring the structure in the batch data. Examining the output structures, we find that our approach only learns additional latent factors when predicting the velocity outputs $\dot{z}$ and $\dot{\theta}$, whereas no additional factors are learned for the position outputs z and θ. These results make sense: the mass of the pole, a single latent factor, directly affects the change in velocity (forces result in accelerations) but does not directly affect the change in position. The values for these factors change monotonically with the mass of the pole, supporting the fact that the discovered factor corresponds to the mass.

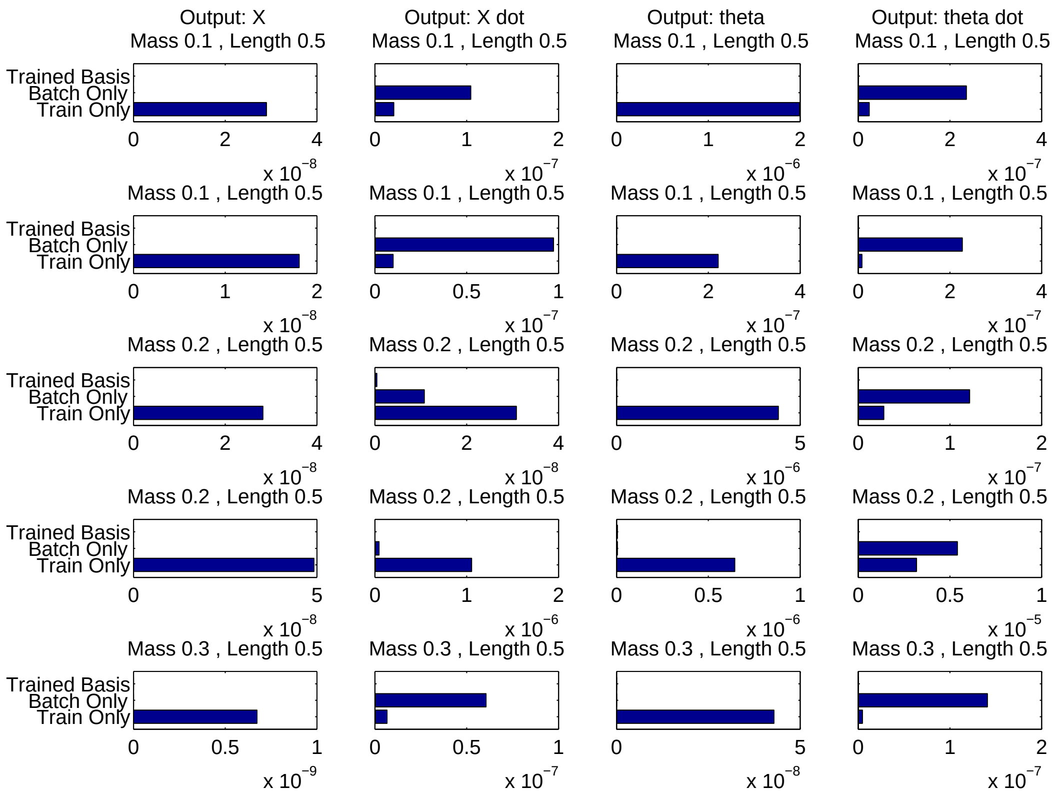

Figure 1: Mean-squared error on predictions when varying the mass of the pole.

Varying the Pole Length We performed a similar test where the agent initially had samples from a pole with mass 0.2 and batch data was collected for pole lengths of 0.4, 0.5, and 0.6. As before, we collected separate training examples for pole lengths of 0.4, 0.45, 0.5, 0.55, and 0.6. In figure 2, we see again that our approach has the lowest mean-squared errors. When varying the pole length, we find that the only case where an additional feature is learned when predicting the angular velocity $\dot{\theta}$. Again, this makes sense from a physics perspective: the change in angular velocity depends on the torque, which is a function of the pole length. However, changing the length of the pole (without changing its mass) does not change the overall mass of the system and thus does not have a direct affect on the velocity of the cart $\dot{x}$. As before, the values associated with the latent parametrization vary monotonically with the pole-length. When predicting the other outputs, the agent still does better than just using the training data or modeling the batch data as a single Gaussian process by combining those two sources of information.

Varying the both Mass and Length Finally, we repeated the same experiment in which the agent received received large batches of data associated with the seven (m, l) settings $\{(.1, .4), (.3, .4), (.15, .45), (.2, .55), (.25, .5), (.3, .4), (.3, .6)\}$ and was asked to predict the change in z, $\dot{z}$, θ , $\dot{\theta}$ on the ten settings $\{(.1, .4), (.1, .45), (.1, .5), (.1, .55), (.1, .6), (.15, .4), (.15, .45), (.15, .5), (.15, .55), (.15, .6)\}$ (note that only three of the settings are in common).

During the batch training, our approach discovered a single additional feature for predicting the change in $\dot{x}$—which makes sense, since only one latent factor, the pole's mass, affects the change in velocity. Three additional factors were discovered for predicting the change in $\dot{\theta}$. While there are only two latent factors in this system, the additive form of our basis regression forces a factor to be learned for the pole's mass, the pole length, and nonlinear interactions between the two factors. No additional factors were learned for the change in position x or the change in angle θ; as described previously, neither the pole's mass nor the pole-length affect these outputs directly.

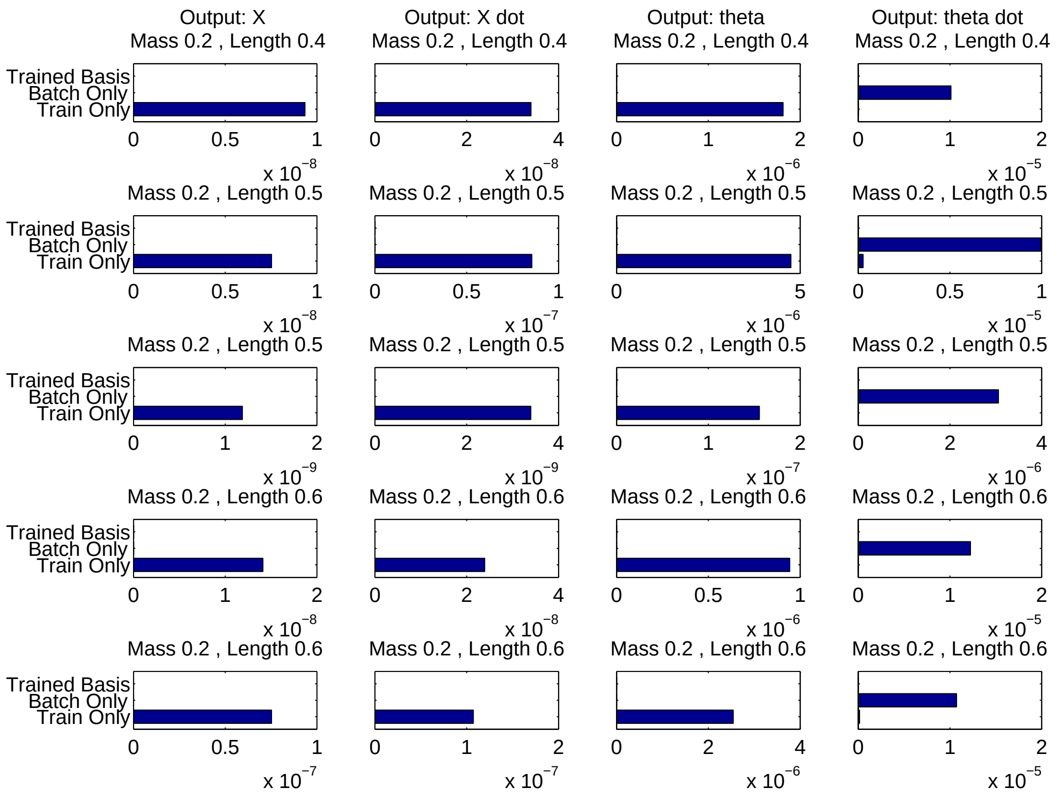

Figure 2: Mean-squared error on predictions when varying the length of the pole.

Figure 3 shows the mean-squared errors for each of the ten runs. We see that when predicting the change in position x or the change in angle θ, using only the small amount of training data from that setting results in large prediction errors. However, because our approach has learned that there are no latent factors, it is able to use all of the batch data as a single Gaussian process to get much lower prediction errors (the errors associated with our approach are the same as the batch only). However, when predicting $\dot{x}$ and $\dot{\theta}$, the latent factors do play a role, and thus using a single Gaussian process is a poor approximation. Errors from the training data alone are still often sizable because only a few data are available. In almost every case, our approach has lower mean-squared errors because we use the training data from a particular setting to leverage structure in the larger batch data set.

4 Discussion and Future Work

When applying machine learning to control, it is often assumed that the agent repeatedly experiences exactly the same task. However, we argue that this assumption is unlikely to survive contact with the real world. A more accurate model of repeated control tasks is that task instances vary, but in limited and specific ways. Accounting for this variance, rather than simply ignoring it, will prove important for building flexible agents capable of adapting to the real world.

One way to view our model is as a compromise between two standard paradigms for control. In reinforcement learning [12], we typically assume that a model of the task we are facing is completely unknown. In planning [8], we typically assume that we are given an exact (though perhaps stochastic) model of our task in advance. The latent parametrization model allows us to think about tasks where we know a great deal about the *class* of task we are facing, but where the variations present in any specific instance still require us to perform learning online.

There has been a great deal of research on transfer for reinforcement learning systems [14], and the most directly related type of transfer is representation transfer [4, 13, 5]. Representation transfer typically learns a set of basis functions sufficient for representation any value function defined in

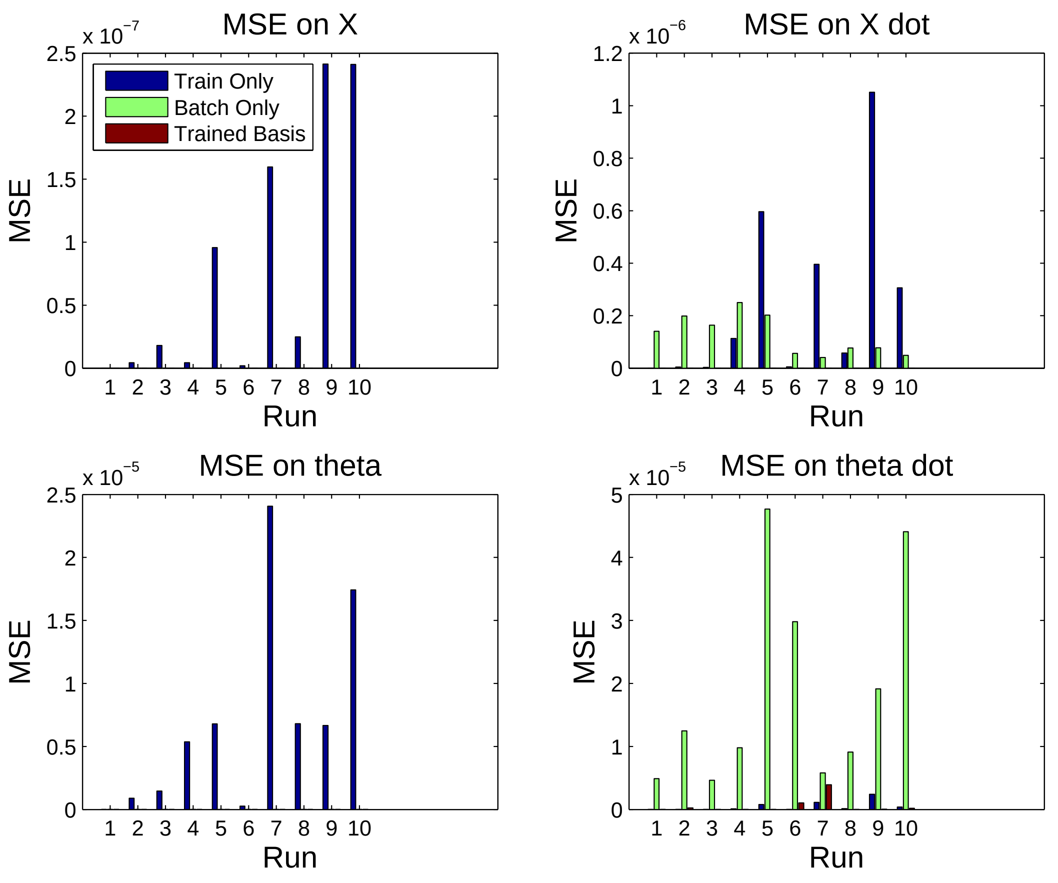

Figure 3: Mean-squared error on predictions when varying both the mass and length of the pole.

a specific state space, or on transfer between two different representations of the same task. By contrast, our work focuses on modeling the dimensions of variation of a family of related tasks.

One way to view the resulting model is as a POMDP, where the hidden states have no dynamics, and are chosen once per episode and then remain constant. This view suggests that, once a good parametrization and model has been obtained, we can apply a POMDP planner to solve the control problem for any specific task instance, either online [6] or by precomputing belief-space policies in advance [16].

We expect that our model will be useful when the family of tasks we are interested in has a parametrization that is small relative to its model, and where knowing this parametrization can make online system identification much more efficient. Many applications fit this scenario—we either know a lot about the dynamics, or can obtain data about the task type in advance. For example, a classical example of applied reinforcement learning is for elevator dispatching [2]. One could imagine a scenario where data from many different buildings are combined to infer the parameters that describes how the users of individual buildings differ in their behavior. Such a parametrization would allow us to deploy an elevator dispatching controller that is immediately good in expectation, but also adapts to its own building's specific needs by updating its belief over the building's latent parameters as it gains experience. Here, the cost of obtaining a significant amount of data in advance to infer the parametrization is easily justified by the resulting rapid improvement in deployed performance. More generally, we may obtain benefits in any scenario where an individual task instance can be thought of as being an individual drawn from a parametrized family of tasks.

In such cases, being able to generalize dynamics from only a few interactions with a new operating regime (using data from many prior interactions with similar systems) is a key step in building controllers that exhibit robust and reliable decision making while gracefully adapt to new situations.

References

[1] J. Asmuth, L. Li, M.L. Littman, A. Nouri, and D. Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of The 25th Conference on Uncertainty in Artificial Intelligence*, pages 501–508, 2009.

[2] R. Crites and A.G. Barto. Improving elevator performance using reinforcement learning. In *Advances in Neural Information Processing Systems 8*, pages 1017–1023. MIT Press, 1996.

[3] B.C. da Silva, G.D. Konidaris, and A.G. Barto. Learning parameterized skills. In *Proceedings of the Twenty Ninth International Conference on Machine Learning*, June 2012.

[4] K. Ferguson and S. Mahadevan. Proto-transfer learning in markov decision processes using spectral methods. In *Proceedings of the ICML Workshop on Structural Knowledge Transfer for Machine Learning*, Pittsburgh PA, June 2006.

[5] E. Ferrante, A. Lazaric, and M. Restelli. Transfer of task representation in reinforcement learning using policy-based proto-value functions. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1329–1332, 2008.

[6] A. Guez, D. Silver, and P. Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. In *Advances in Neural Information Processing Systems 25*, 2012. To appear.

[7] J. Kober, A. Wilhelm, E. Oztop, and J. Peters. Reinforcement learning to adjust parametrized motor primitives to new situations. *Autonomous Robots*, 33(4):316–379, 2012.

[8] S. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[9] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *International Conference in Machine Learning*, pages 697–704, New York, NY, USA, 2006. ACM Press.

[10] S. Ross, B. Chaib-draa, and J. Pineau. Bayesian reinforcement learning in continuous POMDPs with application to robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2008.

[11] D. Silver and J. Veness. Monte-Carlo planning in large POMDPs. In *Advances in Neural Information Processing Systems 23*, pages 2164–2172, 2010.

[12] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[13] M.E. Taylor and P. Stone. Representation transfer for reinforcement learning. In *AAAI 2007 Fall Symposium on Computational Approaches to Representation Change during Learning and Development*, November 2007.

[14] M.E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685, 2009.

[15] Y. W. Teh, M. Seeger, and M. I. Jordan. Semiparametric latent factor models. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*, volume 10, 2005.

[16] Y. Wang, K.S. Won, D. Hsu, and W.S. Lee. Monte carlo bayesian reinforcement learning. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 1135–1142, July 2012.