SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning

Marvin Zhang [*1] **Sharad Vikram** [*2] **Laura Smith** [1] **Pieter Abbeel** [1] **Matthew J. Johnson** [3] **Sergey Levine** [1]

Abstract

Model-based reinforcement learning (RL) has proven to be a data efficient approach for learning control tasks but is difficult to utilize in domains with complex observations such as images. In this paper, we present a method for learning representations that are suitable for iterative model-based policy improvement, even when the underlying dynamical system has complex dynamics and image observations, in that these representations are optimized for inferring simple dynamics and cost models given data from the current policy. This enables a model-based RL method based on the linear-quadratic regulator (LQR) to be used for systems with image observations. We evaluate our approach on a range of robotics tasks, including manipulation with a real-world robotic arm directly from images. We find that our method produces substantially better final performance than other model-based RL methods while being significantly more efficient than model-free RL.

1. Introduction

Model-based reinforcement learning (RL) methods use known or learned models in a variety of ways, such as planning through the model and generating synthetic experience (Sutton, 1990; Kober et al., 2013). On simple, low-dimensional tasks, model-based approaches have demonstrated remarkable data efficiency, learning policies for systems like cart-pole swing-up with under 30 seconds of experience (Deisenroth et al., 2014; Moldovan et al., 2015). However, for more complex domains, one of the main difficulties in applying model-based methods is *modeling bias*: if control or policy learning is performed against an imperfect model, performance in the real world will typically degrade with model inaccuracy (Deisenroth et al., 2014). Many model-based methods rely on accurate forward prediction for planning (Nagabandi et al., 2018; Chua et al., 2018), and for image-based domains, this precludes the use of simple models which will introduce significant modeling bias. However, complex, expressive models must typically be trained on very large datasets, corresponding to days to weeks of data collection, in order to generate accurate forward predictions of images (Finn & Levine, 2017; Pinto & Gupta, 2016; Agrawal et al., 2016).

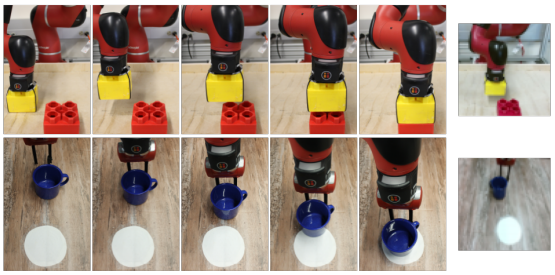


Figure 1. Our method can learn policies for complex manipulation tasks on a real Sawyer robot arm including stacking blocks (top) and pushing a mug onto a coaster (bottom), both from only 64-by-64-by-3 image observations (right), with no additional sensor information, and in one to two hours of interaction time.

How can we use model-based methods to learn from images with similar data efficiency as we have seen in simpler domains? In our work, we focus on removing the need for accurate forward prediction, using what we term *local models methods*. These methods use simple models, typically linear models, to provide gradient directions for local policy improvement, rather than for forward prediction and planning (Todorov & Li, 2005; Levine & Abbeel, 2014). Thus, local model methods circumvent the need for accurate predictive models, but these methods cannot be directly applied to image-based tasks because image dynamics, even locally speaking, are highly non-linear.

Our main contribution is a representation learning and model-based RL procedure, which we term stochastic optimal control with latent representations (SOLAR), that jointly optimizes a latent representation and model such

[*]Equal contribution [1]University of California, Berkeley [2]University of California, San Diego [3]Google. Correspondence to: Marvin Zhang <marvin@eecs.berkeley.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

that inference produces local models that provide good gradient directions for policy improvement. As shown in Figure 1, SOLAR is able to learn policies directly from high-dimensional image observations in several domains, including a real robotic arm stacking blocks and pushing objects with only one to two hours of data collection. To our knowledge, SOLAR is the most efficient RL method for solving real world robotics tasks directly from raw images. We also demonstrate several additional advantages of our method, including the ability to transfer learned models in the multi-task RL setting and the ability to handle sparse reward settings with a set of goal images.

2. Preliminaries

We formalize our setting as a partially observed Markov decision process (POMDP) environment, which is given by the tuple $M = (\mathcal{O}, \mathcal{S}, \mathcal{A}, p, C, f, \rho, T)$. Most prior work in model-based RL assumes the fully observed RL setting where the observation space $\mathcal{O}$ is the same as the state space $\mathcal{S}$ and the observation density function $f(\mathbf{o}|\mathbf{s}) = \delta\{\mathbf{o} = \mathbf{s}\}$ provides the exact state, so we will first discuss this setting. In this setting, the state space $\mathcal{S}$, action space $\mathcal{A}$, and horizon T are known, but the dynamics function $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, cost function $C(\mathbf{s}_t, \mathbf{a}_t)$, and initial state distribution $\rho(\mathbf{s}_1)$ are unknown. RL agents interact with the environment via a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ that chooses an action conditioned on the current state, and the environment responds with the next state, sampled from the dynamics function, and the cost, evaluated through the cost function. The goal of RL is to minimize, with respect to the agent's policy, the expected sum of costs $\eta[\pi] = \mathbb{E}_{\pi,p,\rho}\left[\sum_{t=1}^{T} C(\mathbf{s}_t, \mathbf{a}_t)\right]$. Local model methods iteratively fit dynamics and cost models $\hat{\rho}, \hat{p}, \hat{C}$ to data collected from the current policy in order to optimize $\hat{\eta}[\pi] \triangleq \mathbb{E}_{\pi,\hat{p},\hat{\rho}}\left[\sum_{t=1}^{T} \hat{C}(\mathbf{s}_t, \mathbf{a}_t)\right]$. One particularly tractable and popular model is the linear-quadratic system (LQS), which models the dynamics as time-varying linear-Gaussian (TVLG) and the cost as quadratic, i.e.,

$$\hat{p}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) = \mathcal{N}\left(\mathbf{s}_{t+1} \,\middle|\, \mathbf{F}_t \begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix}, \Sigma_t\right),$$

$$\hat{C}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2}\begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix}^\top \mathbf{C} \begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix} + \mathbf{c}^\top \begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix}.$$

Any deterministic policy operating in an environment with smooth dynamics can be locally modeled with a time-varying LQS (Boyd & Vandenberghe, 2004), while low-entropy stochastic policies are modeled approximately. This makes the time-varying LQS a reasonable local model for many dynamical systems. Furthermore, the optimal maximum-entropy policy $\pi^\star$ under the model is linear-Gaussian state feedback (Jacobson & Mayne, 1970), i.e.,

$$\pi^\star(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}\left(\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t, \mathbf{S}_t\right).$$

We describe how to compute the parameters $\mathbf{K}_t$, $\mathbf{k}_t$, and $\mathbf{S}_t$ in Appendix A. Due to modeling bias, the policy computed through LQR likely will not perform well in the real environment. This is because the model will not be globally correct but rather only valid close to the distribution of the data-collecting policy. One approach to addressing this issue is to use LQR with fitted linear models (LQR-FLM; Levine & Abbeel, 2014), a method which imposes a KL-divergence constraint on the policy update such that the shift in the trajectory distributions before and after the update, which we denote as $\bar{p}(\tau)$ and $p(\tau)$, respectively, is bounded by a step size ϵ. This leads to the constrained optimization

$$\max_{\pi} \hat{\eta}[\pi] \quad \text{s.t.} \quad D_{\mathrm{KL}}(p(\tau)\|\bar{p}(\tau)) \leq \epsilon\,. \tag{1}$$

As shown in Levine & Abbeel (2014), this constrained optimization can be solved by augmenting the cost function to penalize the deviation from the previous policy $\bar{\pi}$, i.e., $\tilde{C}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{\lambda}\hat{C}(\mathbf{s}_t, \mathbf{a}_t) - \log \bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)$. Note that this augmented cost function is still quadratic, since the policy is linear-Gaussian, and thus we can still compute the optimal policy for this cost function in closed form using the LQR procedure. λ is a dual variable that trades off between optimizing the original cost and staying close in distribution to the previous policy, and the weight of this term can be determined through a dual gradient descent procedure.

Methods based on LQR have enjoyed considerable success in a number of control domains, including learning tasks on real robotic systems (Todorov & Li, 2005; Levine et al., 2016). However, most prior work in model-based RL assumes access to a low-dimensional state representation, and this precludes these methods from operating on complex observations such as images. There is some work on lifting this restriction: for example, Watter et al. (2015) and Banijamali et al. (2018) combine LQR-based control with a representation learning scheme based on the variational auto-encoder (VAE; Kingma & Welling, 2014; Rezende et al., 2014) where images are encoded into a learned low-dimensional representation that is used for modeling and control. They demonstrate success on learning several continuous control domains directly from pixel observations. We discuss our method's relationship to this work in Section 6.

3. Learning and Modeling the Latent Space

Representation learning is a promising approach for integrating local models with complex observation spaces like images. What are the desired properties for a learned representation to be useful for local model methods? A simple answer is that local model fitting in a latent space that is low-dimensional and regularized will be more accurate than fitting directly to image observations. Concretely, one approach that satisfies these properties is to embed observations using a standard VAE, where regularization comes in

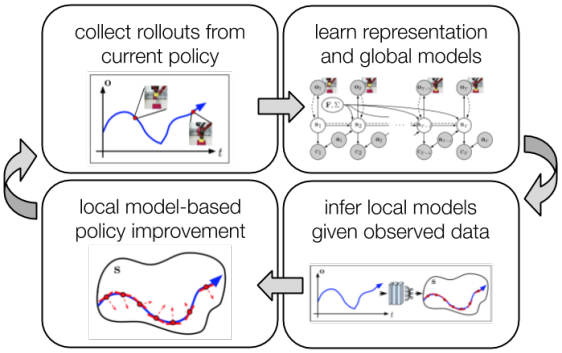


Figure 2. A high-level schematic of our method. We discuss the details of the model and inference procedure in Section 3 and Section 4. We then explain our algorithm in Section 5.

the form of a unit Gaussian prior. However, a VAE representation still may not be amenable to local model fitting since the latent state is not optimized for dynamics and cost modeling. Since we aim to infer local dynamics and cost models in the neighborhood of the observed data, the main property we require from the latent representation is to make this fitting process more accurate for the observed trajectories, thereby reducing modeling bias and enabling a local model method to better improve the policy.

As we discuss in subsection 3.1, in order to make the local model fitting more accurate, especially in the low data regime, we learn global dynamics and cost models on all observed data jointly with the latent representation. Our formulation allows us to directly optimize the latent representation to be amenable for fitting linear dynamics and quadratic cost models, and subsection 3.2 details the learning procedure. Section 4 describes how, using our learned representation and global model as a starting point, we can infer local models that accurately explain the observed data. In this case, the local TVLG dynamics become latent variables in the model. As shown in Figure 2, updating the policy can then be done simply by rolling out a few trajectories, inferring the posterior over the latent TVLG dynamics, and using these dynamics and a local quadratic cost model to improve the policy. This procedure becomes the basis for the SOLAR algorithm which we present in Section 5.

3.1. The Deep Bayesian LQS Model

In our problem setting, we have access to trajectories of the form $[\mathbf{o}_0, \mathbf{a}_0, c_0, \ldots, \mathbf{o}_T, \mathbf{a}_T, c_T]$ sampled from the system using our current policy. We assume this observed data is generated as follows: there is a latent state $\mathbf{s}$ that evolves according to linear-Gaussian dynamics, where the dynamics parameters themselves are stochastic and distributed according to a global prior. At each time step t, the latent state $\mathbf{s}_t$ is used to generate an image observation $\mathbf{o}_t$, and the state and action generate the cost observation c_t. The prior on the dynamics parameters increases the expressivity of the model by removing the assumption that the underlying dynamics are globally linear, since different trajectories may be explained by different samples from the prior. Furthermore, we approximate the observation function with a convolutional neural network, which makes the overall model non-linear. We formalize this generative model as

$$\mathbf{s}_1 \sim \mathcal{N}(0, \mathbf{I}) \ , \tag{2}$$

$$\mathbf{F}, \Sigma \sim \mathcal{MNIW}(\Psi, \nu, \mathbf{M}_0, \mathbf{V}) \ , \tag{3}$$

$$\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t, \mathbf{F}, \Sigma \sim \mathcal{N}\left(\mathbf{F}\begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix}, \Sigma\right) \ , \tag{4}$$

$$\mathbf{o}_t \mid \mathbf{s}_t \sim f_\gamma(\mathbf{s}_t) \ , \tag{5}$$

$$c_t \mid \mathbf{s}_t, \mathbf{a}_t \sim \mathcal{N}\left(\hat{C}(\mathbf{s}_t, \mathbf{a}_t), 1\right) \ . \tag{6}$$

$\mathcal{MNIW}$ denotes the matrix normal inverse-Wishart (MNIW) distribution, which is the conjugate prior for linear-Gaussian dynamics models. Thus, conditioned on transitions from a particular time step, the posterior dynamics distribution $p\left(\mathbf{F}, \Sigma \mid \left\{\mathbf{s}_t^{(i)}, \mathbf{a}_t^{(i)}, \mathbf{s}_{t+1}^{(i)}\right\}_i\right)$ is still MNIW, and we describe in Section 4 how we leverage this conjugacy to infer local linear models using an approximate posterior distribution over the dynamics as a global prior. We refer to $f_\gamma(\mathbf{s})$ as an *observation model* or *decoder*, which is parameterized by neural network weights γ and outputs a Bernoulli distribution over $\mathbf{o}$, which are RGB images.

There are a number of ways to parameterize the quadratic cost model $\hat{C}$, and we detail several options in Appendix B along with an alternate parameterization for sparse human feedback that we discuss in Section 5.

3.2. Joint Model and Representation Learning

We are interested in inferring two distributions of interest, both conditioned on the observations and actions:[1]

1. The posterior distribution over dynamics parameters $p(\mathbf{F}, \Sigma \mid \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$, as this informs our policy update;
2. The posterior distribution over latent trajectories $p(\mathbf{s}_{1:T} \mid \mathbf{o}_{1:T}, \mathbf{a}_{1:T}, \mathbf{F}, \Sigma)$, since we require an estimate of the latent state as the input to our policy.

The subscript $1:T$ denotes an entire trajectory. Both of these distributions are intractable due to the neural network observation model. We instead turn to variational inference which optimizes, with respect to KL-divergence, a variational distribution q in order to approximate a distribution of

[1]Note that we do not condition on the cost observations for simplicity and also because the costs are scalars that contain relatively little information compared to image observations.

interest p. Specifically, we introduce the variational factors

$$q(\mathbf{F}, \Sigma) = \mathcal{MNIW}(\Psi', \nu', \boldsymbol{M}_0', \boldsymbol{V}'),$$
$$q\left(\mathbf{s}_{1:T} \mid \mathbf{F}, \Sigma; \mathbf{o}_{1:T}, \mathbf{a}_{1:T}\right) \propto$$
$$p(\mathbf{s}_1) \prod_{t=1}^{T-1} p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t, \mathbf{F}, \Sigma\right) \prod_{t=1}^{T} \psi(\mathbf{s}_t; \mathbf{o}_t, \phi).$$

$q(\mathbf{F}, \Sigma)$ represents our posterior belief about the system dynamics after observing the collected data, and we also model this distribution as MNIW. We construct the full variational distribution over latent state trajectories as the normalized product of the state dynamics and, borrowing terminology from undirected graphical models, learned *evidence potentials* $\psi(\mathbf{s}_t; \mathbf{o}_t, \phi) = \mathcal{N}(e_\phi(\mathbf{o}_t))$. We refer to $e_\phi(\mathbf{o})$ as a *recognition model* or *encoder*, which is parameterized by neural network weights ϕ and outputs the mean and diagonal covariance of a distribution over $\mathbf{s}$.

To learn the variational parameters, we optimize the evidence lower bound (ELBO), which is given by

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_q\left[\log \frac{p\left(\mathbf{F}, \Sigma, \mathbf{s}_{1:T}, \mathbf{o}_{1:T}, c_{1:T} \mid \mathbf{a}_{1:T}\right)}{q\left(\mathbf{F}, \Sigma, \mathbf{s}_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T}\right)}\right] \\ &= \mathbb{E}_q\left[\sum_{t=1}^{T} \log p(\mathbf{o}_t|\mathbf{s}_t)\right] + \mathbb{E}_q\left[\sum_{t=1}^{T} \log p(c_t|\mathbf{s}_t, \mathbf{a}_t)\right] \\ &\quad - D_{\mathrm{KL}}\left(q\left(\mathbf{F}, \Sigma\right) \| p\left(\mathbf{F}, \Sigma\right)\right) \\ &\quad - \mathbb{E}_q\left[D_{\mathrm{KL}}\left(q\left(\mathbf{s}_{1:T} \mid \mathbf{F}, \Sigma; \mathbf{o}_{1:T}, \mathbf{a}_{1:T}\right) \| \right.\right. \\ &\qquad \left.\left. p\left(\mathbf{s}_{1:T} \mid \mathbf{a}_{1:T}, \mathbf{F}, \Sigma\right)\right)\right].\end{aligned}$$

Johnson et al. (2016) derived an algorithm for optimizing hybrid models with both deep neural networks and probabilistic graphical model (PGM) structure. In fact, our model bears strong resemblance to the LDS SVAE model from their work, though our ultimate goal is to fit local models for model-based policy learning rather than focusing on global models as in their work. We explain the relevant details of the SVAE learning procedure, which we use to learn the neural network parameters γ and ϕ along with the global dynamics and cost models, in Appendix C.

Note that, because the dynamics and cost are learned with samples from the recognition model, we backpropagate the gradients from the cost likelihood and dynamics KL terms through the encoder in order to learn a representation that is better suited to linear dynamics and quadratic cost. Through this, we learn a latent representation that, in addition to being low-dimensional and regularized, is directly optimized for fitting a LQS model on the observed data.

In Figure 3, we depict our generative model using solid lines, and we depict the variational factors and recognition networks using dashed lines. Our method learns two variational distributions: first, a distribution over latent states which is used to provide inputs to the learned policy, and second, a global dynamics model that is used as a prior for inferring local linear dynamics models.

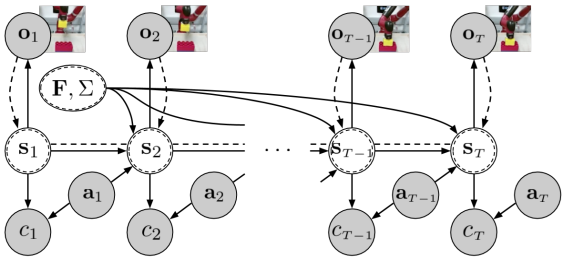


Figure 3. Our generative model, in solid lines, and variational family and recognition network, in dashed lines. In practice, the observations we work with are RGB images, and we use convolutional neural networks for both the recognition and observation models. The distributions for each node are specified in Section 3.

4. Inference and RL in the Latent Space

How can we utilize our learned representation and global models to enable local model methods? As shown in Figure 2, local model methods alternate between collecting batches of data from the current policy and using this data to fit local models and improve the policy. In order to improve the behavior of the local dynamics model fitting, especially in the low data regime, we use our global dynamics model as a prior and fit local dynamics models via posterior inference conditioned on data from the current policy.

For policy improvement, we fit local linear dynamics models separately at every time step, thus we augment the dynamics in our generative model from Equation 3 to instead be separate dynamics parameters $\mathbf{F}_t, \Sigma_t$ at each time step t. We model these parameters as independent samples from the global dynamics model $q(\mathbf{F}, \Sigma)$, and this can be interpreted as an empirical Bayes method, where we use data to estimate the parameters of our priors. In this way, the global dynamics model acts as a prior on the local time-varying dynamics models. In order to then infer the parameters of these local models conditioned on the data from the current policy, we employ a variational expectation-maximization (EM) procedure. The E-step computes $q\left(\mathbf{s}_{1:T} \mid \mathbf{F}_{1:T}, \Sigma_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T}\right)$ given the current local dynamics, which are initialized to the global prior. The M-step optimizes, for each t, $\mathbb{E}[\log q(\mathbf{F}_t, \Sigma_t \mid \mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})]$ with respect to the dynamics parameters, where the expectation is over the latent state distribution from the E-step. We refer readers to Appendix D for complete details.

We additionally fit a local quadratic cost model to the latest batch of data, and this combined with the local linear dynamics models gives us a local latent LQS model. Thus, it is natural to use LQR-based control in order to learn a pol-

Algorithm 1 SOLAR

Input: # iterations K; # trajectories N_{init}, N
Input: model and policy hyperparameters $\xi_{\mathcal{M}}, \xi_{\pi}$
Output: final model $\mathcal{M}$, final policy $\pi^{(K)}$

1: $\pi^{(0)} \leftarrow \text{INITIALIZEPOLICY}(\xi_{\pi})$
2: $\mathcal{D} \leftarrow \text{COLLECTDATA}(N_{\text{init}}, \pi^{(0)})$
3: $\mathcal{M} \leftarrow \text{TRAINMODEL}(\mathcal{D}, \xi_{\mathcal{M}})$ (Section 3)
4: **for** iteration $k \in \{1, \ldots, K\}$ **do**
5: $\quad \{\mathbf{F}_t, \Sigma_t\}_t \leftarrow \text{INFERDYNAMICS}(\mathcal{D}, \mathcal{M})$ (Section 4)
6: $\quad \pi^{(k)} \leftarrow \text{LQR-FLM}(\pi^{(k-1)}, \{\mathbf{F}_t, \Sigma_t\}_t, \mathcal{M})$ (Section 2)
7: $\quad \mathcal{D} \leftarrow \text{COLLECTDATA}(N, \pi^{(k)})$
8: $\quad$ (optional) $\mathcal{M} \leftarrow \text{TRAINMODEL}(\mathcal{D}, \xi_{\mathcal{M}})$
9: **end for**

icy. However, as discussed in Section 2, using vanilla LQR typically leads to undesirable behavior due to modeling bias.

One way to understand the problem is through standard supervised learning analysis, which only guarantees that our local models will be accurate under the distribution of data from the current policy. This directly motivates updating our policy in such a way that the trajectory distribution induced by the new policy does not deviate heavily from the data distribution, and in fact, the update rule proposed by LQR-FLM exactly accomplishes this goal (Levine & Abbeel, 2014). Thus, our policy update method utilizes the same constrained optimization from Equation 1, and we solve this optimization using the same augmented cost function that penalizes deviation from the previous policy.

Note that rolling out our policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ requires computing an estimate of the current latent state $\mathbf{s}_t$. In order to handle partially observable tasks, we estimate the latent state using the history of observations and actions, i.e., $q\left(\mathbf{s}_t \mid \mathbf{F}_{1:t-1}, \Sigma_{1:t-1}; \mathbf{o}_{1:t}, \mathbf{a}_{1:t-1}\right)$, where we condition on the local linear dynamics fit to the latest batch of data. This distribution can be computed using Kalman filtering in the latent space and allows us to handle partial observability by aggregating information that may not be estimable from a single observation, such as system velocity from images.

5. The SOLAR Algorithm

The SOLAR algorithm is presented in Algorithm 1. Lines 1-3 detail the pretraining phase, corresponding to the representation and global model learning described in Section 3, where we collect N_{init} trajectories using a random policy to train the representation, dynamics, and cost model. In our experiments in Section 7, we typically set $N_{\text{init}} \gg N$. In the RL phase, we alternate between inferring dynamics at each time step conditioned on data from the latest policy as described in Section 4 (line 5), performing the LQR-FLM update described in Section 2 given the inferred dynamics (line 6), collecting N trajectories using the updated policy (line 7), and optionally fine-tuning the model on the new data (line 8).[2] The model hyperparameters $\xi_{\mathcal{M}}$ include number of iterations, learning rates, and minibatch size, and the policy hyperparameters ξ_{π} include the policy update KL constraint ϵ and the initial random variance.

We evaluate SOLAR in Section 7 in several RL settings involving continuous control including manipulation tasks on a real Sawyer robot. Beyond our method's performance on these tasks, however, we can derive several other significant advantages from our representation and PGM learning. As we detail in the rest of this section, these advantages include transfer in the multi-task RL setting and handling sparse reward settings using an augmented graphical model.

5.1. Transferring Representations and Models

In the scenario where the dynamics are unknown, LQR-based methods are typically used in a "trajectory-centric" fashion where the distributions over initial conditions and goal conditions are low variance (Levine & Abbeel, 2014; Chebotar et al., 2017). We similarly test our method in such settings in Section 7, e.g., learning Lego block stacking where the top block starts in a set position and the bottom block is fixed to the table. In the more general case where we may wish to handle several different conditions, we can learn a policy for each condition, however this may require significant amounts of data if there are many conditions.

However, one significant advantage of representation and model learning over alternative approaches, such as model-free RL, is the potential for transferring knowledge across multiple tasks where the underlying system dynamics do not change (Lesort et al., 2018). Here, we consider each condition to be a separate task, and given a task distribution, we first sample various tasks and learn our model from Section 3 using random data from these tasks. We show in Section 7 that this "base model" can then be directly transferred to new tasks within the distribution, essentially removing the pretraining phase and dramatically speeding up learning for the Sawyer Lego block stacking domain.

5.2. Learning from Sparse Rewards

Reward functions can often be hard to specify for complex tasks in the real world, and in particular they may require highly instrumented setups such as motion capture when operating from image observations. In these settings, sparse feedback is often easier to specify as it can come directly from a human labeler. Because we incorporate PGM ma-

[2] In our experiments, we found that fine-tuning the model did not improve final performance, though this step may be more important for environments where exploration is more difficult.

chinery in our learned latent representation, it is straightforward for SOLAR to handle alternate forms of supervision simply by augmenting our generative model to reflect how the new supervision is given. Specifically, we extend our cost model to the sparse reward setting by assuming that we observe a binary signal f_t based on the policy performance, rather than costs c_t, and then modeling f_t as a Bernoulli random variable with probability given by

$$p(f_t = 1 \mid \mathbf{s}_t, \mathbf{a}_t) \propto \exp\left\{-\hat{C}(\mathbf{s}_t, \mathbf{a}_t)\right\}$$

Concretely, in our experiments, f_t is generated by a human that only provides $f_t = 1$ when the task is solved. This setup is reminiscent of Fu et al. (2018), though our goal is not to classify expert data from policy data. Learning $\hat{C}$ from observing f_t amounts to logistic regression, and afterwards we can use $\hat{C}$ as before in order to perform control and policy learning. Note that we can still backpropagate gradients through the encoder in order to learn a representation that is more amenable to predicting f_t. In Section 7, we use this method to solve a pushing task for which providing rewards is difficult without motion capture, and instead we use sparse human feedback and a set of goal images to specify the desired outcome. We provide the implementation details for this experiment in Appendix E.

6. Related Work

Utilizing representation learning within model-based RL has been studied in a number of previous works (Lesort et al., 2018), including using embeddings for state aggregation (Singh et al., 1994), dimensionality reduction (Nouri & Littman, 2010), self-organizing maps (Smith, 2002), value prediction (Oh et al., 2017), and deep auto-encoders (Lange & Riedmiller, 2010; Higgins et al., 2017). Among these works, deep spatial auto-encoders (DSAE; Finn et al., 2016) and embed to control (E2C; Watter et al., 2015; Banijamali et al., 2018) are the most closely related to our work, in that they consider local model methods combined with representation learning. The key difference in our work is that, rather than using a learning objective for reconstruction and forward prediction, our objective is more suited for local model methods by directly encouraging learning representations where fitting local models accurately explains the observed data. We also do not assume a known cost function, goal state, or access to the underlying system state as in DSAE and E2C, making SOLAR applicable even when the underlying states and cost function are unknown.[3]

Subsequent to our work, Hafner et al. (2018) formulate a representation and model learning method for image-based continuous control tasks that is used in conjunction with model-predictive control (MPC), which plans H time steps ahead using the model, executes an action based on this plan, and then re-plans after receiving the next observation. We compare to a baseline that uses MPC in Section 7, and we empirically demonstrate the relative strengths of SOLAR and MPC, showing that SOLAR can overcome the short-horizon bias that afflicts MPC. We also compare to robust locally-linear controllable embedding (RCE; Banijamali et al., 2018), an improved version of E2C, and we find that our approach tends to produce better empirical results.

[3]These methods may be extended to unknown underlying states and cost functions, though the authors do not experiment with this and it is unclear how well these approaches would generalize.

7. Experiments

We aim to answer the following through our experiments:

1. What benefits do we derive by utilizing model-based RL and representation learning in general?
2. How does SOLAR compare to similar methods in terms of solving image-based control tasks?
3. Can we utilize SOLAR to solve image-based control tasks on a real robotic system?

To answer 1, we compare SOLAR to PPO (Schulman et al., 2017), a state-of-the-art model-free RL method, and LQR-FLM with no representation learning. For the real world tasks, we also compare to deep visual foresight (DVF; Ebert et al., 2018), a state-of-the-art model-based method for images which does not use representation learning.

To answer 2, we compare to RCE (Banijamali et al., 2018), which as discussed earlier is an improved version of E2C (Watter et al., 2015). We also set up an "VAE ablation" of SOLAR where we replace our representation learning scheme with a standard VAE. Finally, we consider an "MPC baseline" where we train neural network dynamics and cost models jointly with a latent representation and then use MPC with these models. Details regarding each of the comparisons are in Appendix F.

To answer 3, we evaluate SOLAR on a block stacking task and a pushing task on a Sawyer robot arm as shown in Figure 1. Videos of the learned policies are available at `https://sites.google.com/view/icml19solar`.

7.1. Experimental Tasks

We set up simulated image-based robotic domains as well as manipulation tasks on a real Sawyer robotic arm, as shown in Figure 4. Details regarding task setup and training hyperparameters are provided in Appendix E.

2D navigation. Our 2-dimensional navigation task is similar to Watter et al. (2015) and Banijamali et al. (2018) where an agent controls its velocity in a bounded planar system to reach a specified target. However, we make this task harder

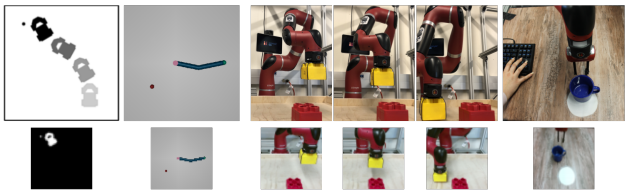
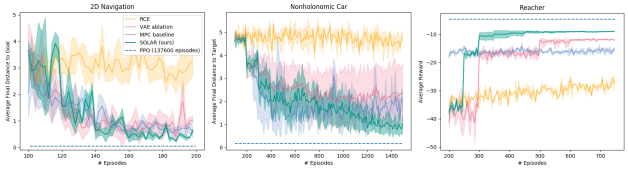


Figure 4. Illustrations of the environments we test on in the top row with example image observations in the bottow row. Left to right: visualizing a trajectory in the nonholonomic car environment, with the target denoted by the black dot; an illustration of the 2-DoF reacher environment, with the target denoted by the red dot; the different tasks that we test for block stacking, where the rightmost task is the most difficult as the policy must learn to first lift the yellow block before stacking it; a depiction of our pushing setup, where a human provides the sparse reward that indicates whether the robot successfully pushed the mug onto the coaster.



(a) (b) (c)

Figure 5. Full size versions of these plots are available on the project website. (a): Our method, the MPC baseline, and the VAE ablation consistently solve 2D navigation with a randomized goal, whereas RCE is unable to make progress. The final performance of PPO is plotted as the dashed line, though PPO requires 1000 times more samples than our method to reach this performance. (b): On the nonholonomic car, both our method and the MPC baseline are able to reach the goal, though the VAE ablation is less consistent across seeds and RCE once again is unsuccessful at the task. PPO requires over 25 times more episodes than our method to learn a successful policy. (c): On reacher, we perform worse than PPO but use about 40 times fewer episodes. RCE fails to learn at all, and the VAE ablation and MPC baseline are noticeably worse than our method. Here we plot reward, so higher is better.

by randomizing the goal every episode rather than fixing it to the bottom right. Observations consist of two 32-by-32 images showing the positions of the agent and goal.

Nonholonomic car. The nonholonomic car starts in the bottom right of the 2-dimensional space and controls its acceleration and steering velocity in order to reach the target in the top left. We use 64-by-64 images as the observation.

Reacher. We experiment with the reacher environment from OpenAI Gym (Brockman et al., 2016), where a 2-DoF arm in a 2-dimensional plane has to reach a fixed target denoted by a red dot. For observations, we directly use 64-by-64-by-3 images of the rendered environment, which provides a top-down view of the reacher and target.

Sawyer Lego block stacking. To demonstrate a challenging domain in the real world, we use our method to learn Lego block stacking with a real 7-DoF Sawyer robotic arm. The observations are 64-by-64-by-3 images from a camera pointed at the robot, and the controller only receives images as the observation without joint angles or other information. As shown in Figure 4, we define different block stacking tasks as different initial positions of the Sawyer arm.

Sawyer pushing. We also experiment with the Sawyer arm learning to push a mug onto a white coaster, where we again use 64-by-64-by-3 images with no auxiliary information. Furthermore, we set up this task with only sparse binary rewards that indicate whether the mug is on top of the coaster, which are provided by a human labeler.

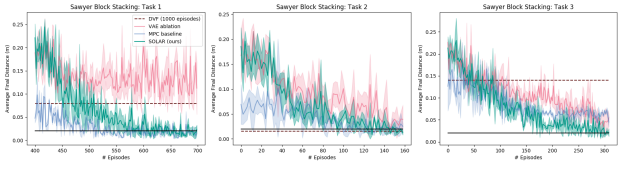
7.2. Comparisons to Prior Work

As shown in Figure 5, we compare to prior methods only on the simulated domains as these methods have not been shown to solve real-world image-based domains with reasonable data efficiency. On the 2D navigation task, our method, the VAE ablation, and the MPC baseline are able to learn very quickly, converging to high-performing policies in 200 episodes. However, these policies still exhibit some "jittery" behavior due to modeling bias, especially for the VAE ablation, whereas PPO learns an extremely accurate policy that continues to improve the longer we train. This gain in asymptotic performance is typical of model-free methods over model-based methods, however achieving this performance requires two to three orders of magnitude more samples. We present log-scale plots that illustrate the full learning progress of PPO in Appendix G.

LQR-FLM from pixels fails to learn anything meaningful, and its performance does not improve over the initial policy. In fact, LQR-FLM does not make progress on any of the tasks, and for the sake of clarity in the plots, we omit these results. Similarly, despite extensive tuning and using code directly from the original authors, we were unable to get RCE to learn a good model for our 2D navigation task, and thus the learned policy also does not improve over the initial policy. RCE did not learn successful policies for any of the other tasks that we experiment with, though in Appendix G, we show that RCE can indeed learn the easier fixed-target 2D navigation task from prior work.

On the nonholonomic car, our method and the MPC baseline are able to learn with about 1500 episodes of experience, whereas the VAE ablation's performance is less consistent. PPO eventually learns a successful policy for this task that performs better than our method, however it requires over 25 times more data to reach this performance.

Our method is outperformed by the final PPO policy on the reacher task, however, PPO requires about 40 times more



(a) (b) (c)

Figure 6. Our method consistently solves all block stacking tasks. The MPC baseline learns very quickly on the two easier tasks since it can plan through the pretrained model, however, due to the short-horizon planning, it performs significantly worse on the hard task on the right where the block starts on the table. The VAE ablation performs well on the easiest task in the middle but is unsuccessful on the two harder tasks. DVF makes progress for each task but ultimately is not as data efficient as SOLAR. The black solid line at 0.02m denotes successful stacking.

Method	Final Distance to Goal (cm)	Episodes per Seed
DVF (Ebert et al., 2018)	4.50 ± 2.60	280
SOLAR (ours)	1.85 ± 0.86	250

Table 1. Sawyer Pushing with Sparse Rewards

Figure 7. Visualizing example end states from rolling out our policy after 200 (top), 230 (middle) and 260 (bottom) trajectories.

data to learn. The VAE ablation and MPC baseline also make progress toward the target, though the performance is noticeably worse than our method. MPC often has better initial behavior than LQR-FLM as it uses the pretrained models right away for planning, highlighting one benefit of planning-based methods, however the MPC baseline barely improves past this behavior. Forward prediction with this learned model deteriorates quickly as the horizon increases, which makes long-horizon planning impossible. MPC is thus limited to short-horizon planning, and this limitation has been noted in prior work (Nagabandi et al., 2018; Feinberg et al., 2018). SOLAR does not suffer from this as we do not use our models for forward prediction.

Our open-source implementation of SOLAR is available at `https://github.com/sharadmv/parasol`.

7.3. Analysis of Real Robot Results

The real-world Lego block stacking results are shown in Figure 6. Our method is successful on all tasks, where we define success as achieving an average distance of 0.02m which generally corresponds to successful stacking, whereas the VAE ablation is only successful on the easiest task in the middle plot. The MPC baseline again starts off better and learns more quickly on the two easier tasks. However, MPC is again limited to short-horizon planning, which causes it to fail on the most difficult task in the right plot as it simply greedily reduces the distance between the two blocks rather than lifting the block off the table. We can solve each block stacking task using about two hours of robot interaction time, though the x-axes in the plots show that we further reduce the total data requirements by about a factor of two by pretraining and transferring a shared representation and global model as described in Section 5.

As a comparison to a state-of-the-art model-based method that has been successful in real-world image-based domains, we evaluate DVF (Ebert et al., 2018), which learns pixel space models and does not utilize representation learning. We find that this method can make progress but ultimately is not able to solve the two harder tasks even with more data than what we use for our method and even with a much smaller model. This highlights our method's data efficiency, as we use about two hours of robot data compared to days or weeks of data as in this prior work.

Finally, on the real-world pushing task, despite the additional challenge of sparse rewards, our method learns a successful policy in about an hour of interaction time as detailed in Table 1 and visualized in Figure 7. DVF performs worse than our method with a comparable amount of data, again even when using a downsized model. Videos depicting the learning process for both of the real-world tasks, as well as full size versions of the plots and learning curves, are available at `https://sites.google.com/view/icml19solar`.

8. Discussion

We presented SOLAR, a model-based RL algorithm that is capable of learning policies in a data-efficient manner directly from raw high-dimensional image observations. The key insights in SOLAR involve learning latent representations where simple models are more accurate and utilizing PGM structure to infer dynamics from data conditioned on observed trajectories. Our experimental results demonstrate that SOLAR is competitive in sample efficiency, while exhibiting superior final policy performance, compared to other model-based methods. SOLAR is also significantly more data-efficient compared to model-free RL methods, especially when transferring previously learned representations and models. We show that SOLAR can learn complex real-world robotic manipulation tasks with only image observations in one to two hours of interaction time.

Our model is designed for and tested on continuous action domains, and extending our model to discrete actions would necessitate some type of learned action representation. This is intriguing also as a potential mechanism for further reducing modeling bias. Certain systems such as dexterous hands and tensegrity robots not only exhibit complex state spaces but also complex action spaces (Zhu et al., 2018; Andrychowicz et al., 2018; Zhang et al., 2017), and learning simpler action representations that can potentially capture high-level behavior, such as manipulation or locomotion primitives, is an exciting line of future work.

Acknowledgments. MZ is supported by an NDSEG fellowship. SV is supported by NSF grant CNS1446912. This work was supported by the NSF, through IIS-1614653, and computational resource donations from Amazon.

References

Agrawal, P., Nair, A., Abbeel, P., Malik, J., and Levine, S. Learning to poke by poking: Experiential learning of intuitive physics. In *NIPS*, 2016.

Andrychowicz, M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., and Zaremba, W. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Banijamali, E., Shu, R., Ghavamzadeh, M., Bui, H., and Ghodsi, A. Robust locally-linear controllable embedding. In *AISTATS*, 2018.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge University Press, 2004.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. OpenAI gym. *arXiv preprint arXiv:1606.01540*, 2016.

Camacho, E. and Alba, C. *Model Predictive Control*. Springer Science and Business Media, 2013.

Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G., Schaal, S., and Levine, S. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *ICML*, 2017.

Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NIPS*, 2018.

Deisenroth, M., Fox, D., and Rasmussen, C. Gaussian processes for data-efficient learning in robotics and control. *PAMI*, 2014.

Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A., and Levine, S. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Feinberg, V., Wan, A., Stoica, I., Jordan, M., Gonzalez, J., and Levine, S. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Finn, C. and Levine, S. Deep visual foresight for planning robot motion. In *ICRA*, 2017.

Finn, C., Tan, X., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. Deep spatial autoencoders for visuomotor learning. In *ICRA*, 2016.

Fu, J., Singh, A., Ghosh, D., Yang, L., and Levine, S. Variational inverse control with events: A general framework for data-driven reward definition. In *NIPS*, 2018.

Fujimoto, S., van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: Improving zero-shot transfer in reinforcement learning. In *ICML*, 2017.

Hoffman, M., Blei, D., Wang, C., and Paisley, J. Stochastic variational inference. *JMLR*, 2013.

Jacobson, D. and Mayne, D. *Differential Dynamic Programming*. American Elsevier, 1970.

Johnson, M., Duvenaud, D., Wiltschko, A., Datta, S., and Adams, R. Composing graphical models with neural networks for structured representations and fast inference. In *NIPS*, 2016.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Kingma, D. and Welling, M. Auto-encoding variational Bayes. In *ICLR*, 2014.

Kober, J., Bagnell, J., and Peters, J. Reinforcement learning in robotics: A survey. *IJRR*, 2013.

Lange, S. and Riedmiller, M. Deep auto-encoder neural networks in reinforcement learning. In *IJCNN*, 2010.

Lesort, T., Díaz-Rodríguez, N., Goudou, J., and Filliat, D. State representation learning for control: An overview. *Neural Networks*, 2018.

Levine, S. and Abbeel, P. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2014.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

Moldovan, T., Levine, S., Jordan, M., and Abbeel, P. Optimism-driven exploration for nonlinear systems. In *ICRA*, 2015.

Nagabandi, A., Kahn, G., Fearing, R., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

Nouri, A. and Littman, M. Dimension reduction and its application to model-based exploration in continuous spaces. *Machine Learning*, 2010.

Oh, J., Singh, S., and Lee, H. Value prediction network. In *NIPS*, 2017.

Pinto, L. and Gupta, A. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *ICRA*, 2016.

Rezende, D., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Singh, S., Jaakkola, T., and Jordan, M. Reinforcement learning with soft state aggregation. In *NIPS*, 1994.

Smith, A. Applications of the self-organizing map to reinforcement learning. *Neural Networks*, 2002.

Sutton, R. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, 1990.

Tassa, Y., Erez, T., and Todorov, E. Synthesis and stabilization of complex behaviors. In *IROS*, 2012.

Todorov, E. and Li, W. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *ACC*, 2005.

Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. In *NIPS*, 2015.

Winn, J. and Bishop, C. Variational message passing. *JMLR*, 2005.

Zhang, M., Geng, X., Bruce, J., Caluwaerts, K., Vespignani, M., SunSpiral, V., Abbeel, P., and Levine, S. Deep reinforcement learning for tensegrity robot locomotion. In *ICRA*, 2017.

Zhu, H., Gupta, A., Rajeswaran, A., Levine, S., and Kumar, V. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. *arXiv preprint arXiv:1810.06045*, 2018.

A. Policy Learning Details

Given a TVLG dynamics model and quadratic cost approximation, we can approximate our Q and value functions to second order with the following dynamic programming updates, which proceed from the last time step $t = T$ to the first step $t = 1$:

$$Q_{\mathbf{s},t} = c_{\mathbf{s},t} + \mathbf{F}_{\mathbf{s},t}^\top V_{\mathbf{s},t+1}\,, \quad Q_{\mathbf{ss},t} = c_{\mathbf{ss},t} + \mathbf{F}_{\mathbf{s},t}^\top V_{\mathbf{ss},t+1}\mathbf{F}_{\mathbf{s},t}\,,$$
$$Q_{\mathbf{a},t} = c_{\mathbf{a},t} + \mathbf{F}_{\mathbf{a},t}^\top V_{\mathbf{s},t+1}\,, \quad Q_{\mathbf{aa},t} = c_{\mathbf{aa},t} + \mathbf{F}_{\mathbf{a},t}^\top V_{\mathbf{ss},t+1}\mathbf{F}_{\mathbf{a},t}\,,$$
$$Q_{\mathbf{sa},t} = c_{\mathbf{sa},t} + \mathbf{F}_{\mathbf{s},t}^\top V_{\mathbf{ss},t+1}\mathbf{F}_{\mathbf{a},t}\,,$$
$$V_{\mathbf{s},t} = Q_{\mathbf{s},t} - Q_{\mathbf{sa},t}Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{a},t}\,,$$
$$V_{\mathbf{ss},t} = Q_{\mathbf{ss},t} - Q_{\mathbf{sa},t}Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{as},t}\,.$$

It can be shown (e.g., by Tassa et al. (2012)) that the action $\mathbf{a}_t$ that minimizes the second-order approximation of the Q-function at every time step t is given by

$$\mathbf{a}_t = -Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{as},t}\mathbf{s}_t - Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{a},t}\,.$$

This action is a linear function of the state $\mathbf{s}_t$, thus we can construct an optimal linear policy by setting $\mathbf{K}_t = -Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{as},t}$ and $\mathbf{k}_t = -Q_{\mathbf{aa},t}^{-1}Q_{\mathbf{a},t}$. We can also show that the maximum-entropy policy that minimizes the approximate Q-function is given by

$$\pi^\star(\mathbf{a}_t|\mathbf{s}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{s}_t + \mathbf{k}_t, Q_{\mathbf{aa},t}).$$

Furthermore, as in Levine & Abbeel (2014), we can impose a constraint on the total KL-divergence between the old and new trajectory distributions induced by the policies through an augmented cost function $\tilde{C}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{\lambda}\hat{C}(\mathbf{s}_t, \mathbf{a}_t) - \log\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)$, where solving for λ via dual gradient descent can yield an exact solution to a KL-constrained LQR problem.

B. Parameterizing the Cost Model

The simplest choice that we consider for parameterizing the cost model is as a full quadratic function of the state and action, i.e., $\hat{C}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2}\mathbf{s}_t^\top\mathbf{C}\mathbf{s}_t + \mathbf{c}^\top\mathbf{s}_t + \alpha\|\mathbf{a}_t\|_2^2 + b$ where we assume that the action-dependent part of the cost – i.e., α – is known, and we impose no restrictions on the learned parameters $\mathbf{C}$ and $\mathbf{c}$. This is our default option due to its simplicity and the added benefit that fitting this model locally can be done in closed form through least-squares quadratic regression on the observed states. However, another option we consider is to choose $\hat{C}(\mathbf{s}_t, \mathbf{a}_t) = \frac{1}{2}\mathbf{s}_t^\top\mathbf{L}\mathbf{L}^\top\mathbf{s}_t + \mathbf{c}^\top\mathbf{s}_t + \alpha\|\mathbf{a}_t\|_2^2 + b$. $\mathbf{L}$ is a lower-triangular matrix with non-negative diagonal entries, and thus by constructing our cost matrix as $\mathbf{C} = \mathbf{L}\mathbf{L}^\top$ we guarantee that the learned cost matrix is positive semidefinite, which can improve the behavior of the policy update.

In general, in this work, we consider quadratic parameterizations of the cost model since we wish to build a LQS model. However, in general it may be possible to use non-quadratic but twice-differentiable cost models, such as a neural network model, and compute local quadratic cost models using a second-order Taylor approximation as in Levine & Abbeel (2014). We also do not assume access to a goal observation, though if provided with such information we can construct a quadratic cost function that penalizes distance to this goal in the learned latent space, as in Finn et al. (2016) and Watter et al. (2015).

C. The SVAE Algorithm

Johnson et al. (2016) build off of Hoffman et al. (2013) and Winn & Bishop (2005), who show that, for conjugate exponential models, the variational model parameters can be updated using natural gradients of the form

$$\tilde{\nabla}_\omega\mathcal{L} = \omega^0 + B\mathbb{E}_q\left[t_{\mathbf{F},\Sigma}(\mathbf{F},\Sigma)\right] - \omega\,, \tag{7}$$

Where ω denotes the MNIW parameters of the variational factors on $\mathbf{F}, \Sigma$, B is the number of minibatches in the dataset, ω^0 is the parameter for the prior distribution $p(\mathbf{F}, \Sigma)$, and $t_{\mathbf{F},\Sigma}(\mathbf{F}, \Sigma)$ is the sufficient statistic function for $p(\mathbf{F}, \Sigma)$. Thus, we can use this equation to compute the natural gradient update for ω, whereas for γ, ϕ, and the parameters of the cost model, we use stochastic gradient updates on Monte Carlo estimates of the ELBO, specifically using the Adam optimizer (Kingma & Ba, 2015). This leads to two simultaneous optimizations, and their learning rates are treated as separate hyperparameters. We have found 10^{-4} and 10^{-3} to be good default settings for the natural gradient step size and stochastic gradient step size, respectively.

D. Fitting the Local Dynamics Model

In the pretraining phase described in Section 3, we are learning the following sets of parameters from observed trajectories:

1. The parameters of the variational posterior over global dynamics $q_{\text{global}}(\mathbf{F}, \Sigma)$;
2. The weights of the encoder and decoder networks $f_\gamma(\mathbf{s})$ and $e_\phi(\mathbf{o})$;
3. The parameters of the cost function $\hat{C}(\mathbf{s}, \mathbf{a})$.

In the RL phase described in Section 4, after learning the representation and global models, we fit local linear-Gaussian dynamics models to additional trajectories. The conjugacy of the Bayesian LQS model enables a computationally efficient expectation-maximization procedure to learn the local dynamics. We assume the same graphical model as in Equation 2 to Equation 6 except we modify Equation 3 and

Equation 4 to be

$$\mathbf{F}_t, \Sigma_t \sim p(\mathbf{F}_t, \Sigma_t) \triangleq q_{\text{global}}(\mathbf{F}, \Sigma)\,,$$
$$\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{F}_t, \Sigma_t \sim \mathcal{N}\left(\mathbf{F}_t \begin{bmatrix}\mathbf{s}_t \\ \mathbf{a}_t\end{bmatrix}, \Sigma_t\right)\,.$$

The model assumes that the TVLG dynamics are independent samples from our global dynamics, followed by a deep Bayesian LDS to generate trajectories. This is similar to the globally trained model, with the exception that we explicitly assume time-varying dynamics.

Now suppose we have collected a set of trajectories of the form $[\mathbf{o}_0, \mathbf{a}_0, c_0, \ldots, \mathbf{o}_T, \mathbf{a}_T, c_T]$ and aim to fit a local dynamics model. We use variational inference to approximate the posterior distributions by setting up the variational factors

1. $q(\mathbf{s}_{1:T}|\mathbf{F}_{1:T}, \Sigma_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$, which approximates the posterior distribution $p(\mathbf{s}_{1:T}|\mathbf{o}_{1:T}, \mathbf{a}_{1:T}, \mathbf{F}_{1:T}, \Sigma_{1:T})$;
2. $q(\mathbf{F}_t, \Sigma_t)$, which approximates the posterior distribution $p(\mathbf{F}_t, \Sigma_t|\mathbf{s}_{1:T}, \mathbf{a}_{1:T})$

The ELBO under these variational factors is:

$$\begin{aligned}\mathcal{L} = \mathbb{E}_q\big[&\sum_t^T \log p(\mathbf{o}_t|\mathbf{s}_t) \\ &- \text{KL}\left(q(\mathbf{s}_{1:T}) \| p(\mathbf{s}_{1:T}|\mathbf{a}_{1:T}, \mathbf{F}_{1:T}, \Sigma_{1:T})\right) \\ &- \sum_t^{T-1} \text{KL}\left(q(\mathbf{F}_t, \Sigma_t) \| p(\mathbf{F}_t, \Sigma_t)\right) \big]\end{aligned}$$

We use variational EM to alternatively optimize $q(\mathbf{s}_{1:T}|\mathbf{F}_{1:T}, \Sigma_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$ and $q(\mathbf{F}_t, \Sigma_t)$. Using evidence potentials $\psi(\mathbf{s}_t; \mathbf{o}_t, \phi)$ output by the recognition network $e_\phi(\mathbf{o}_t)$, both of these optimizations can be done in closed form. Specifically, the optimal $q(\mathbf{s}_{1:T}|\mathbf{F}_{1:T}, \Sigma_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$ is computed via Kalman smoothing using evidence potentials from the recognition network, and the optimal $q(\mathbf{F}_t, \Sigma_t)$ can be computed via Bayesian linear regression using expected sufficient statistics from $q(\mathbf{s}_{1:T}|\mathbf{F}_{1:T}, \Sigma_{1:T}; \mathbf{o}_{1:T}, \mathbf{a}_{1:T})$.

E. Experiment Setup

2D navigation. Our recognition model architecture for the 2D navigation domain consists of two convolution layers with 2-by-2 filters and 32 channels each, with no pooling layers and ReLU non-linearities, followed by another convolution with 2-by-2 filters and 2 channels. The output of the last convolution layer is fed into a fully-connected layer which then outputs a Gaussian distribution with diagonal covariance. Our observation model consists of FC hidden layers with 256 ReLU activations, and the last layer outputs a categorical distribution over pixels. We initially collect 100 episodes which we use to train our model, and for every subsequent RL iteration we collect 10 episodes. The cost function we use is the sum of the L^2-norm squared of the distance to the target and the commanded action, with weights of 1 and 0.001, respectively.

As discussed in Section 7, we modify the 2D navigation task from Watter et al. (2015) and Banijamali et al. (2018) to randomize the location of the target every episode, and we set this location uniformly at random between -2.8 and 2.8 for both the x and y coordinates, as coordinates outside of $[-3, 3]$ are not visible in the image. We similarly randomize the initial position of the agent. In this setup, we use two 32-by-32 images as the observation, one with the location of the agent and the other with the location of the target, and in the fixed-target version of the task we only use one 32-by-32 image.

Nonholonomic car. The nonholonomic car domain consists of 64-by-64 image observations. Our recognition model is a convolutional neural network with four convolutional layers with 4-by-4 filters with 4 channels each, and the first two convolution layers are followed by a ReLU non-linearity. The output of the last convolutional layer is fed into three FC ReLU layers of width 2048, 512, and 128, respectively. Our final layer outputs a Gaussian distribution with dimension 8. Our observation model consists of four FC ReLU layers of width 256, 512, 1024, and 2048, respectively, followed by a Bernoulli distribution layer that models the image. For this domain, we collect 100 episodes initially to train our model, and then for RL we collect 100 episodes per iteration. The cost function we use is the sum of the L^2-norm squared of the distance from the center of the car to the target and the commanded action, with weights of 1 and 0.001, respectively.

Reacher. The reacher domain consists of 64-by-64-by-3 image observations. Our recognition model consists of three convolutional layers with 7-by-7, 5-by-5, and 3-by-3 filters with 64, 32 and 8 channels respectively. The first convolutional layer is followed by a ReLU non-linearity. The output of the last convolutional layer is fed into an FC ReLU layer of width 256, which outputs a Gaussian distribution with dimension 10. Our observation model consists of one FC ReLU layers of width 512, followed by three deconvolutional layers with the reverse order of filters and channels as the recognition model. This is followed by a Bernoulli distribution layer that models each image. We collect 200 episodes initially to train our model, and then for RL we collect 100 episodes per iteration. The cost function we use is the sum of the L^2-norm of the distance from the fingertip to the target and the L^2-norm squared of the commanded action, which is the negative of the reward function as defined in Gym.

Sawyer Lego block stacking. The image-based Sawyer block-stacking domain consists of 64-by-64-by-3 image observations. The policy outputs velocities on the end effector in order to control the robot. Our recognition model is a convolutional neural network with the following architecture: a 5-by-5 filter convolutional layer with 16 channels followed by two convolutional layers using 5-by-5 filters with 32 channels each. The convolutional layers are followed by ReLU activations leading to a 12 dimensional Gaussian distribution layer. Our observation model consists of a FC ReLU layer of width 128 feeding into three deconvolutional layers, the first with 5-by-5 filters with 16 channels and the last two of 6-by-6 filters with 8 channels each. These are followed by a final Bernoulli distribution layer.

For this domain, we collect 400 episodes initially to train our model and 10 per iteration thereafter. Note that this pretraining data is collected only once across solving all of the tasks that we test on. The cost function is the cubed root of the L^2-norm of the displacement vector between the end-effector and the target in 3D-space.

Sawyer pushing. The image-based Sawyer pushing domain also operates on 64-by-64-by-3 image observations. Our recognition and observation models are the same as those used in the block-stacking domain. The dynamics model is learned by a network with two FC ReLU layers of width 128 followed by a 12 dimensional Gaussian distribution layer. The cost model is learned jointly with the representation and dynamics by optimizing the ELBO, which with regards to the cost corresponds to logistic regression on the observed sparse reward using a sampled latent state as the input. We collect 200 episodes to train our model and 20 per iteration for RL.

During the RL phase, the human supervisor uses keyboard input to provide the sparse reward signal to the learning algorithm, indicating whether or not the mug was successfully pushed onto the coaster. In practice, for simplicity, we label the last five images of the trajectory as either 0 or 1 depending on whether or not the keyboard was pressed at any time during the trajectory, as for this task a successful push is typically reflected in the end state. In order to overcome the exploration problem and provide a diverse dataset for pretraining the cost model, we manually collect 180 "goal images" where the mug is on the coaster and the robot arm is in various locations.

F. Implementation of Comparisons

PPO. We use the open source implementation of PPO (named "PPO2") from the OpenAI Baselines project: `https://github.com/openai/baselines`. We write OpenAI gym wrappers for our simulated environments in order to test PPO on our simulated tasks.

LQR-FLM. We implement LQR-FLM based on the open-source implementation from the Guided Policy Search project: `https://github.com/cbfinn/gps`. The only modification to the LQR-FLM algorithm that we make is to handle unknown cost functions by fitting a quadratic cost model to data from the current policy.

DVF. We train a video prediction model using the open source Stochastic Adversarial Video Prediction project: `https://github.com/alexlee-gk/video_prediction`. To define the task, we specify the location of a pixel whose movement to a specified goal location indicates success. The cost function is then the predicted probability of successfully moving the selected pixel to the goal. We then use MPC, specifically the cross-entropy method (CEM) for offline planning: we sample sequences of actions from a Gaussian, predict the corresponding sequence of images using the video prediction model, evaluate the cost of the imagined trajectory with the cost model, and refit the parameters of the Gaussian to the best predicted action sequences. This iterative process eventually outputs an action sequence to perform in the real world in order to try and solve the task.

RCE. We use model learning code directly from the authors of RCE (Banijamali et al., 2018), though this code is not publicly available and to our knowledge there are no open source implementations of RCE or E2C (Watter et al., 2015) that are able to reproduce the results from the respective papers. In addition to LQR-based control, we also experiment with MPC with neural network dynamics and cost models in the learned latent representation. In our experiments, we report the best results using either of these control methods.

VAE ablation. In the VAE ablation, we replace our representation and global models with a standard VAE (Kingma & Welling, 2014; Rezende et al., 2014), which imposes a unit Gaussian prior on the latent representation. Because we cannot infer local dynamics as described in Section 4, we instead use a GMM dynamics prior that is trained on all data as described by Levine et al. (2016). After fitting a local quadratic cost model, we again have a local LQS model that we can use in conjunction with an LQR-FLM policy update.

MPC baseline. (MPC) involves planning H time steps ahead using a dynamics and cost model, executing an action based on this plan, and then re-planning after receiving the next observation (Camacho & Alba, 2013). Recently, MPC has proven to be a successful control method when combined with neural network dynamics models, where many trajectories are sampled using the model and then the first action corresponding to the best imagined trajectory is executed (Nagabandi et al., 2018; Chua et al., 2018). Similar to LQR-FLM, we can extend MPC to handle image-based domains by learning dynamics and cost models within a

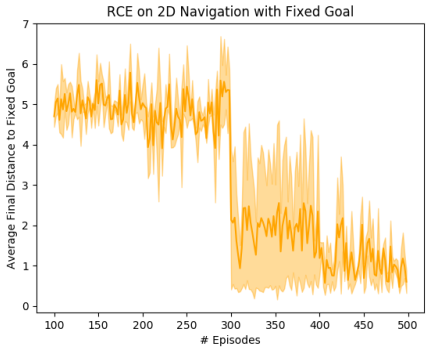


Figure 8. On 2D navigation with the goal fixed to the bottom right, RCE is able to successfully learn a policy for navigating to the goal.

learned latent representation. As MPC does not require an LQS model, we can instead utilize neural network dynamics and cost models which are more expressive.

G. Additional Experiments

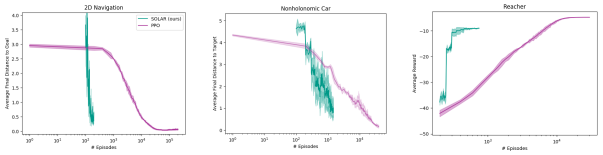
G.1. RCE on Fixed-Target 2D Navigation

As mentioned in Section 7, RCE was unable to make progress for the 2D navigation task, though we were able to get more successful results by fixing the position of the goal to the bottom right as is done in the image-based 2D navigation task considered in E2C (Watter et al., 2015) and RCE (Banijamali et al., 2018). Figure 8 details this experiment, which we ran for three random seeds and report the mean and standard deviation of the average final distance to the goal as a function of the number of training episodes. This indicates that RCE can indeed solve some tasks from image observations, though we were unable to use RCE succesfully on any of the tasks we consider.

G.2. Full Learning Progress of PPO

In Figure 9 we include the plots for the simulated tasks comparing SOLAR and PPO. Note that the x-axis is on a log scale, i.e., though our method is sometimes worse in final policy performance, we use one to three orders of magnitude fewer samples. This demonstrates our method's sample efficiency compared to PPO, while being able to solve complex image-based domains that are difficult for model-based methods.

PPO is an on-policy model-free RL method, and typically off-policy methods exhibit better sample efficiency (Fujimoto et al., 2018; Haarnoja et al., 2018). We use PPO in our comparisons because on-policy methods are typically easier to tune, at the cost of being less efficient, and the complexity of our image-based environments poses a major challenge



(a) (b) (c)

Figure 9. (a) Comparison of our method to PPO on the 2D navigation task presented in the paper. Our method uses roughly three orders of magnitude fewer samples to solve the task compared to PPO. (b) On the car from images task, our method achieves slightly worse performance than PPO though with about 25 times fewer samples. (c) Comparison of our method to PPO for the reacher task. Our method achieves worse final performance but uses about 40 times fewer samples than these methods.

for all RL methods. Specifically, we also compared to TD3 (Fujimoto et al., 2018), and we were unable to train successful policies despite extensive hyperparameter tuning. We also note that, to our knowledge, TD3 has never been tested on image-based domains.