Imagined Value Gradients: Model-Based Policy Optimization with Transferable Latent Dynamics Models

Arunkumar Byravan[2*] **Jost Tobias Springenberg**[1] **Abbas Abdolmaleki**[1] **Roland Hafner**[1]

Michael Neunert[1] **Thomas Lampe**[1] **Noah Siegel**[1] **Nicolas Heess**[1]

Martin Riedmiller[1] [1] DeepMind, London [2] University of Washington

Abstract: Humans are masters at quickly learning many complex tasks, relying on an approximate understanding of the dynamics of their environments. In much the same way, we would like our learning agents to quickly adapt to new tasks. In this paper, we explore how model-based Reinforcement Learning (RL) can facilitate transfer to new tasks. We develop an algorithm that learns an action-conditional, predictive model of expected future observations, rewards and values from which a policy can be derived by following the gradient of the estimated value along imagined trajectories. We show how robust policy optimization can be achieved in robot manipulation tasks even with approximate models that are learned directly from vision and proprioception. We evaluate the efficacy of our approach in a transfer learning scenario, re-using previously learned models on tasks with different reward structures and visual distractors, and show a significant improvement in learning speed compared to strong off-policy baselines. Videos with results can be found at https://sites.google.com/view/ivg-corl19

Keywords: RL, Model-Based RL, Transfer Learning, Visuomotor Control

1 Introduction

The last decade has seen significant progress in reinforcement learning. The field has matured to a state where RL can solve challenging simulated motor control problems [1, 2] or games [3, 4] even from high-dimensional observations such as images [5, 6, 7]. Off-policy model-free algorithms have become workable for high-dimensional continuous action spaces [8, 9, 10], and have improved in robustness and data-efficiency allowing experiments directly on robotics hardware [11, 12, 13, 14].

Model-free techniques directly learn a policy (and value function) from environment interactions. Their simplicity, generality and versatility has been a major factor behind their recent successes. Yet, these techniques do not entirely satisfy the intuition that a learning agent should be able to acquire approximate knowledge of the dynamics of its environment in a manner that is independent of any particular task and such that it is easily applicable to new tasks, more easily than task-specific objects such as policies. It is this intuition and as well as the desire to further improve the sample-efficiency and transferability of solutions that has driven much of the recent research in model-based RL.

A growing body of literature is concerned with learning dynamics models for physical control problems [15, 16], including approaches that learn latent models from images [17, 18, 19]. Although some approaches excel in data efficiency [15, 20], in general, model-based methods have not yet achieved the robustness of model-free techniques; and they still struggle with model inaccuracies and long planning horizons. When learned dynamic models are combined with policy learning [21, 22, 23, 24, 25] the advantages over purely model-free techniques can be less clear.

In this work we develop an approach that uses model-based RL to learn a stochastic parametric policy in domains with high-dimensional observation spaces such as images. We build on Stochastic

[*] Work done during an internship at DeepMind.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

Value Gradients (SVG) [9] which allow us to compute low-variance policy gradients by directly differentiating a model-based value function. We extend this work in two ways: Firstly, we develop a latent state space model that allows us to predict expected future reward and value as a function of the current policy even when the true low-dimensional system state is not directly observed. Secondly, rather than using the model only for credit assignment along observed trajectories we use it directly to produce "imagined" rollouts and show that stable policy learning can still be achieved.

We apply our algorithm to several challenging long-horizon vision-based manipulation tasks (e.g. lifting and stacking) in simulation and demonstrate the following: Our model-based approach (a) is as robust and achieves the same asymptotic performance as competitive model-free baselines, and (b) in several cases it significantly improves data-efficiency. (c) It can effectively transfer the learned model to novel tasks with different reward distributions or visual distractors, leading to a dramatic gain in data-efficiency in such settings. (d) It is particularly effective in a multi-task setup where the models learned on multiple tasks learn faster and generalize better to new tasks, successfully solving problems that cannot be learned in isolation.

2 Related Work

Model-free RL: Model-free RL has recently been applied to many challenging problems including robotics [26]. These successes were helped by advances in algorithms suitable for the use of powerful function approximators [5, 8, 27, 10, 3, 28], also enabling the use of RL in domains with high-dimensional observation spaces [5, 4, 28]. However, model-free methods can still struggle in the low data regime and their primary objects of interest, policies and value functions are task specific and can thus be difficult to transfer.

Model-based RL: On the other end of the spectrum, model-based methods learn a model of the environment and use a planner to generate actions. One of the successes in this area is work by Deisenroth et al. [15] who learn uncertainty aware policies using Gaussian-Process models and showed impressive data efficiency on multiple low-dimensional tasks. Using deep neural networks, model based RL has been extended to vision-based problems such as Atari [6] and complex manipulation tasks [20, 16]. Other work has studied learned vision-based latent representations for RL [17, 29, 19, 30, 18]: Such low-dimensional representations can provide feature spaces in which learning and planning can proceed significantly faster compared to learning directly from images although model inaccuracies, sparse rewards, and long-horizons remain challenging.

Model-free + Model-based: Several papers have focused on combining the strengths of model-free and model-based RL. The work on Dyna [21] was among the first. It integrates an action model and model-based imagined rollouts with policy learning, interleaving planning, learning and execution in a tight loop. Recent work has further explored the use of imagined rollouts generated with learned models for accelerating learning of model-free policies [22, 23] – or indirectly speeding up learning via model-based value estimates [24, 31]. These approaches propose different means to handle model approximation errors, such as using short rollouts to avoid cascading model errors [24], uncertainty estimation through model ensembles [31] and using the rollouts as policy conditioning variables only [23]. Contrasting these methods, Stochastic Value Gradients (SVG) [9] re-evaluates rollouts with a learned model from off-policy data, accelerating learning through value gradients back-propagated through time via a learned model. In this work, we extend SVG to use imagined rollouts in latent spaces, accounting for model approximation errors via gradient averaging.

There has also been work on combining latent space models and model-free RL. DeepMDP [32], MERLIN [25], CRAR [33] and VPN [34] combine representation learning with policy optimization via auxiliary losses such as observation reconstruction [25], predicting the next latent state [32] or future values [34]. Along these lines, we learn a latent representation that can be used for predicting (expected) future latent states, observations, rewards and values conditioned on the observed actions.

Transfer: One argument for model-based RL is the potential of transfer to tasks in the same environment. Sutton et al. [21] discuss early examples of this type of transfer on simple problems. Recently, Francois et al. [33] showed encouraging results with a vision-based RL agent on 2D labyrinth tasks. Nagabandi et al. [35, 36] meta-learn predictive dynamics models to enable a model-based agent to rapidly adapt to changes in its environment. Other work on transfer in RL has focused on learning reusable skills e.g. in the form of embedding spaces [37, 38, 39], successor representations [40], transferable priors [41] or meta-policies [42, 43]. As they learn "behaviors" rather than dynamics models, they are in a sense orthogonal and complementary to the ideas presented in this work.

3 Background

We are interested in solving motor control problems such as robotic manipulation tasks from vision. This setup can be formalized as a partially observed Markov decision process (a POMDP) with observation $\mathbf{o}^t \in R^{N_O}$, states $\mathbf{s}^t \in R^{N_S}$, actions $\mathbf{a}^t \in R^{N_A}$ transition probabilities $p(\mathbf{s}^{t+1}|\mathbf{s}^t, \mathbf{a}^t)$, and reward function $r^t = r(\mathbf{s}^t, \mathbf{a}^t)$. Let $\tau_{\leq t} = (\mathbf{o}^{1:t}, \mathbf{a}^{1:t-1})$ be the sequence of observations and actions in a trajectory up to decision point t. The optimal policy and the value function at time step t in this setting are a function of the posterior of the system state $p(\mathbf{s}_t|\tau_{\leq t})$ given the interaction history. However, this posterior is usually intractable and many reinforcement learning approaches resort, instead, to directly optimizing a parametric policy that is a function of the history $\tau_{\leq t}$, i.e. they consider policies of the form $\pi(\mathbf{a}_t|\tau_{\leq t})$. Thus, they aim to maximize the sum of discounted rewards: $J(\theta) = \mathbb{E}_{p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T}|\tau_{\leq t})}\left[\sum_{t=0}^{T} \gamma^t r^t\right]$, with discount factor $\gamma \in [0, 1)$, where the trajectory distribution is assumed to decompose into transition and action probabilities as $p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T}|\tau_{\leq t}) = p(\mathbf{s}^t|\tau_{\leq t}) \prod_{t'\geq t}^{T-1} \pi(\mathbf{a}^{t'}|\tau_{\leq t'}) p(\mathbf{s}^{t'+1}|\mathbf{s}^{t'}, \mathbf{a}^{t'}) p(\mathbf{o}^{t'+1}|\mathbf{s}^{t'+1})$. This allows us to define the value of an observed trajectory prefix as

$$V^\pi(\tau_{\leq t}) = \mathbb{E}_{p_\pi(\mathbf{s}^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T}|\tau_{\leq t})}\left[\sum_{k=t}^{T} \gamma^{k-t} r^t\right], \tag{1}$$

where, below, we consider the infinite horizon case" $\lim_{T\to\infty}$.

4 An Action-Conditional Expectation Model of Observations and Rewards

Naively, a model-based evaluation of (1) would require an accurate action-conditional model of future observations and rewards given a a partial trajectory $\tau_{\leq t}$. Especially in high-dimensional observation spaces such a model can be difficult to learn. Instead we consider an approximate model suitable for weakly partially observed domains with limited stochasticity. We develop a latent state-space model whose latent state at time step t is optimized to represent a sufficient statistic of the interaction history $\tau_{\leq t}$. The model is trained to predict expectations of future observations and reward by approximately modeling the evolution of the summary statistic via a deterministic transition model. We express the policy as a function of the latent state and the model allows us to construct a surrogate model $\tilde{V}$ that expresses the value $V^\pi(\tau_{\leq t})$ as a recursive function of the policy, the deterministic transition function, and a learned approximation to the reward. This surrogate model can be used to compute approximate policy gradients.

More specifically, we let $\mathbf{h}^t = f_{\text{enc}}(\mathbf{o}^{1:t}; \phi)$ be a determistic mapping (with parameters ϕ) extracting a summary statistic from a history of observations[2]; and we let $\mathbf{h}^{t+1} \triangleq f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t; \phi)$ denote a latent transition function. The assumption that the latent dynamics are well described by a deterministic transition function is our primary simplification. We further define the approximate reward function based on latent states as $\hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi) = \hat{r}(f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t); \phi)$ which we calculate by chaining the transition model with a reward predictor. Finally, we denote with $\pi_\theta(\mathbf{a}|\mathbf{h})$ a stochastic policy, parameterized by θ. Using these definitions we construct an approximation of the expectation in Eq. (1) as $V^\pi(\tau_{\leq t}) \approx \tilde{V}^\pi(\mathbf{h}^t) = \mathbb{E}_{\pi_\theta}\left[\sum_{k=t}^{\infty} \gamma^{k-t} \hat{r}^k | \mathbf{h}^t = \mathbf{h}\right]$ or, written recursively:

$$V^\pi(\tau_{\leq t}) \approx \tilde{V}^\pi(\mathbf{h}^t) = \int \left[\hat{r}(\mathbf{h}^t, \mathbf{a}; \phi) + \gamma \tilde{V}^\pi(\mathbf{h}^{t+1}) \middle| \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}; \phi)\right] \pi_\theta(\mathbf{a}|\mathbf{h}^t) \mathrm{d}\mathbf{a}, \tag{2}$$

where the initial latent state is given as $\mathbf{h}^t = f_{\text{enc}}(\mathbf{o}^{1:t}; \phi)$. In the following we will describe our approach in two steps: We first describe how to learn the action-conditional model for f_{enc}, f_{trans}, $\hat{r}$, and thus $\tilde{V}$ in Sec. 4.1. We then explain in Sec. 5 how the model can be used to optimize the policy.

4.1 Model Learning

We need to estimate all quantities comprising Equation (2); i.e., we need to estimate the following parametric functions (for brevity we use a single set of parameters ϕ for all model parameters):

Encode $f_{enc}(\mathbf{o}^{1:t}; \phi) = \mathbf{h}^t \approx \mathbb{E}_{p(\mathbf{s}|\mathbf{o}^{1:t})}[\mathbf{s}^t]$ **Transition** $f_{trans}(\mathbf{h}^t, \mathbf{a}; \phi) = \mathbf{h}^{t+1}$
Decode $f_{dec}(\mathbf{h}^t; \phi) \approx \mathbb{E}_{p(\mathbf{o}|\mathbf{s}^t, \tau_{\leq t})}[\mathbf{o}^t]$ **Value** $\hat{V}^\pi(\mathbf{h}^t; \phi) \approx V^\pi(\tau_{\leq t})$
Reward $\hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi) = \hat{r}(f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t); \phi) \approx \mathbb{E}_{p(\mathbf{s}^t|\tau_{\leq t}, \mathbf{a}^t)}[r(\mathbf{s}^t, \mathbf{a}^t)]$

[2]We drop the dependency on actions here, assuming $\mathbf{s}$ is retrievable from a history of observations only.

The **Encoder** maps a history of observations $\mathbf{o}^{1:t}$ to a summary statistic or latent state $\mathbf{h}^t$, via a recurrent neural network. Recurrence allows us to handle partially observable settings (eg: occlusion). The **Transition** function predicts the next latent state $\mathbf{h}^{t+1}$ given the current state $\mathbf{h}^t$ and action $\mathbf{a}$, evolving dynamics in the low-dimensional latent space. The **Decoder** maps the latent $\mathbf{h}^t$ back to an expected observation $\mathbf{o}^t$ and is primarily used as self-supervision for training the encoder [25]. The **Value** function, predicts the sum of expected rewards (the value) as a function of a latent $\mathbf{h}^t$. Lastly, the **Reward** function predicts the immediate, expected reward for a given latent state-action pair. Please refer to Section E of the supplementary for additional details on the model architecture.

For the model-based value function $\tilde{V}$ in Eq. (2) to form a good approximation of the true value V^π (and its gradient) we train the model on trajectories collected while interacting with the environment. The main approximation of our approach is the assumption that the evolution of the latent state is well modeled by a deterministic transition function. In partially observed and stochastic environments this is not guaranteed. For the relevant quantities to be well approximated despite this simplification we employ a number of losses that satisfy the following desiderata: i) we want to ensure that $\mathbf{h}^t$ is a sufficient statistic of the history $\mathbf{o}^{1:t}$ and the system thus Markov in $\mathbf{h}$; ii) we want to minimize the discrepancy between the predicted and observed evolution of the latent state $\mathbf{h}$; iii) given a latent state $\mathbf{h}^t$ we want to accurately predict the expected reward and value (of policy π) at time step $t+k$ after executing some action sequence $\mathbf{a}^{t:t+k}$.

Let $\mathcal{B}$ denote a set of trajectory data collected while executing some behavior policy $\mu(\mathbf{a}|\mathbf{h})$. Let us define the full model loss after an initial "burn-in" of H steps (to ensure the encoder has sufficient information) as $\mathcal{L}^N = \mathbb{E}_{p_\pi,\mu}[\sum_{t=H+1}^{H+N-1} \mathcal{L}_e(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) | f_{\text{trans}}, f_{\text{enc}}]$ where we approximate the expectation wrt. $p_\pi(r^{t:T}, \mathbf{a}^{t:T}, \mathbf{o}^{t+1:T} | \tau_{\leq t})$ with samples from $\mathcal{B}$,

$$\mathcal{L}^{\mathcal{N}} \approx \mathbb{E}_{\tau\sim\mathcal{B}}\left[\sum_{t=H+1}^{H+N-1} \mathcal{L}_e(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) \Big| \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \mathbf{a}^t; \phi), \mathbf{h}^H = f_{\text{enc}}(\mathbf{o}^{1:H}; \phi)\right], \quad (3)$$

with $\tau := (\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N})$ and where the per example loss is defined as $\mathcal{L}_e = \mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) + \alpha\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) + \beta\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1})$. α and β are coefficients that determine the relative contribution of the loss components. The per example transition model loss is given as

$$\mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) = \|f_{\text{dec}}(\mathbf{h}^t; \phi) - \mathbf{o}^t\|_2^2 + \zeta\|f_{\text{enc}}(\mathbf{o}^{1:t}; \phi) - \mathbf{h}^t\|_2^2, \quad (4)$$

where the first term measures the error between the observations $\mathbf{o}$ and reconstructions from the open-loop latent state predictions ($\mathbf{h}^{t>H}$) and the second term enforces consistency between the latent state representation from the encoder f_{enc} and the predictions from the transition model f_{trans}; this encourages the latent state to stay close to encodings of observed trajectories thus addressing points i) and ii) above. Here ζ is a coefficient that weights the two loss terms and the reward loss is

$$\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) = \|\hat{r}(\mathbf{h}^t, \mathbf{a}^t; \phi) - r^t\|_2^2, \quad (5)$$

where the value-loss is given by the, importance weighted, squared Bellman error

$$\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) = \frac{\pi(\mathbf{a}^t|\mathbf{h}^t)}{\mu(\mathbf{a}^t|\mathbf{h}^t)}\left(r^t + \gamma\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) - \hat{V}^\pi(\mathbf{h}^t; \phi)\right)^2, \quad (6)$$

where the next state value $\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi)$ is calculated via a "target network", whose parameters ψ are periodically copied from ϕ, to stabilize training (see e.g. [5] for a discussion). In practice we use a v-trace target [7]; see discussion in the supplementary. Note that both loss terms are evaluated for *predicted* latent states with gradients flowing backwards through the transition model and eventually the encoder, which addresses point iii). We encourage the reader to consult to Section C & Figure 1 in the supplementary material for additional details and a schematic.

5 Imagined Value Gradients in Latent Spaces

Given a model, we optimize a parametric policy $\pi_\theta(\mathbf{a}|\mathbf{h})$ by maximizing the N-step surrogate value function which is a recursive composition of the policy, the transition, reward and value function:

$$\tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{\mathbf{a}^k\sim\pi}\left[\gamma^N\hat{V}^\pi(\mathbf{h}^{t+N}; \phi) + \sum_{k=t}^{t+N-1} \gamma^{k-t}\hat{r}(\mathbf{h}^k, \mathbf{a}^k) \Big| \mathbf{h}^{k+1} = f_{\text{trans}}(\mathbf{h}^k, \mathbf{a}^k)\right], \quad (7)$$

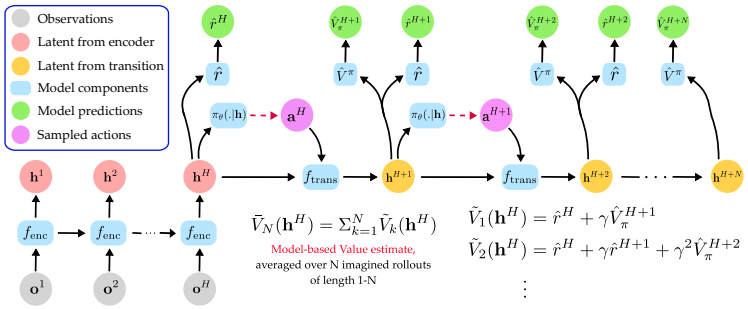


Figure 1: Imagined policy gradient computation. Given a history H of observations from $\mathcal{B}$, we encode a latent state $\mathbf{h}^H$, followed by an "imagined" rollout of length N – using sequence of sampled actions $\mathbf{a}^{t>H}$ and f_{trans}. This leads to imagined states $\mathbf{h}^{t>H}$ with corresponding value and reward estimates. We average cumulative rewards over N horizons – computing the estimate from Eq. (9).

This N-step value can be computed by performing an "imagined" rollout in the latent state-space using our model (see Fig.1). It can be maximized by gradient ascent, exploiting the so called "value gradient" $\nabla_\theta \tilde{V}_N(\mathbf{h}^t)$ [9]; which can often be computed recursively, taking advantage of the reparameterization trick [44, 45] for sampling from π_θ and calculating analytic gradients via back-propagation backwards through time. We can express the policy as a deterministic function $\pi_\theta(\mathbf{h}^t, \epsilon)$ that transforms a sample ϵ from a canonical noise distribution $p(\epsilon)$ into a sample from $\pi_\theta(\mathbf{a}|\mathbf{h})$. In the following we will consider Gaussian policy distributions, i.e. $\pi_\theta(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{h}), \sigma^2_\theta(\mathbf{h}))$, for which the reparameterization is given as $\pi_\theta(\mathbf{h}, \epsilon) = \mu_\theta(\mathbf{h}) + \epsilon\sigma_\theta(\mathbf{h})$, with $p(\epsilon) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix. We can then calculate the gradient $\nabla_\theta \tilde{V}_N(\mathbf{h}^t)$ for any state as

$$\nabla_\theta \tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{p(\epsilon)}\left[\left(\nabla_{\mathbf{a}} \hat{r}\left(\mathbf{h}^t, \mathbf{a}\right) + \gamma \nabla_{\mathbf{h}'} \tilde{V}_{N-1}(\mathbf{h}') \nabla_{\mathbf{a}} f_{\text{trans}}\left(\mathbf{h}^t, \mathbf{a}\right)\right) \nabla_\theta \pi_\theta(\mathbf{h}^t, \epsilon) + \gamma \nabla_\theta \tilde{V}_{N-1}(\mathbf{h}')\right] \tag{8}$$

where $\mathbf{h}' := \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t, \pi_\theta(\mathbf{h}^t, \epsilon))$ and we dropped the dependencies on ϕ for brevity. The value gradient $\nabla_{\mathbf{h}'} \tilde{V}_{N-1}(\mathbf{h}')$ wrt. a state $\mathbf{h}'$ is defined recursively and provided in Eq. (3) in the supplementary material. The case $\nabla_\theta \tilde{V}_1(\mathbf{h}^{t+N})$ is established by assuming the policy is fixed in all steps after N; i.e. bootstrapping with $\nabla_{\mathbf{h}^t} \tilde{V}_0(\mathbf{h}^t) = \nabla_{\mathbf{h}^t} \hat{V}^\pi(\mathbf{h}^t; \phi)$. To calculate the gradient, only an initial state $\mathbf{h}^t$ (encoded from $\mathbf{o}^{1:t} \sim \mathcal{B}$) is required (see Fig.1). Eq. (8) computes N policy gradient contributions for the encoded state $\mathbf{h}^t$ as well as for the imagined states $\mathbf{h}^{t+1} \ldots \mathbf{h}^{t+N-1}$. This ensures that the policy can be evaluated on either kind of latent state. Our derivation is analogous to SVG [9]; but using imagined latent states and assuming a deterministic transition model.

5.1 Stable Regularized Policy Optimization

We make the following observations regarding the use of value gradients in practice: First, we can obtain different gradient estimators by varying N. For small N we obtain a more biased value gradient – due to the reliance on bootstrapping – that changes slowly (i.e. it changes at the speed of convergence for $\hat{V}^\pi(\mathbf{h}; \phi)$). For large N, less bias, and faster learning could be achieved if model and reward predictors are accurate, but the estimate can be affected by modelling errors. As a compromise, we found that averaging gradient estimates obtained with different-length rollouts worked well in practice $\nabla_\theta \bar{V}_N(\mathbf{h}) = \frac{1}{N} \sum_k^N \nabla_\theta \tilde{V}_k(\mathbf{h})$. We refer to the supplementary for details.

Second, even with this averaged gradient, optimization is prone to exploiting modelling errors (in the transition dynamics and reward/value estimates) – see e.g. [18] for a discussion. To counteract this, we employ relative-entropy (KL) regularization. Similar to existing policy optimization methods [27, 10, 12] we augment the estimated reward with a KL penalty, yielding $\hat{r}_{\text{KL}}(\mathbf{h}, \mathbf{a}, \pi_\theta) = \hat{r}(\mathbf{h}, \mathbf{a}) + \lambda \log \frac{\pi_\theta(\mathbf{a}|\mathbf{h})}{p(\mathbf{a}|\mathbf{h})}$, where $p(\mathbf{a}|\mathbf{h})$ is a the prior action probability (we use $p(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ throughout) and λ is a cost multiplier. Replacing $\hat{r}$ in Equation (8) with $\hat{r}_{\text{KL}}$ – noting that $\hat{r}_{\text{KL}}$ is differentiable wrt. θ – results in the regularized value gradient $\nabla_\theta \tilde{V}^{\text{KL}}_N(\mathbf{h})$. Analogously, we obtain a compatible value $\tilde{V}^{\text{KL}}_0(\mathbf{h})$ by replacing r with r_{KL} in Equation (6). The total derivative estimate we use then is

$$\nabla_\theta \mathbb{E}_{p_\pi}\left[\bar{V}^{\text{KL}}_N(\mathbf{h}^t)\right] \approx \mathbb{E}_{\mathbf{o}^{1:t} \sim \mathcal{B}}\left[\frac{1}{N} \sum_{k=1}^N \nabla_\theta \tilde{V}^{\text{KL}}_k(f_{\text{enc}}(\mathbf{o}^{1:t}))\right], \tag{9}$$

where we use batches from the replay to optimize the policy on all visited states – using Eq. (9) in combination with Adam [46]. Please refer to Algorithm (1) in the supplementary for a description.

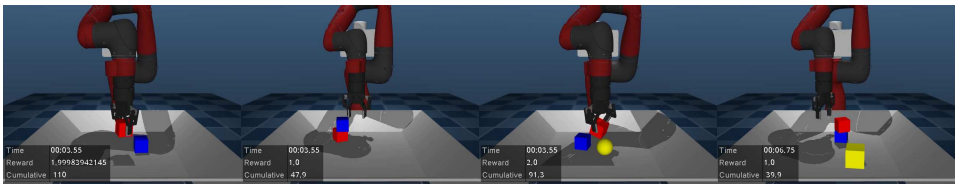


Figure 2: *Left*: **Lift-R** task - robot lifts the red block. *Center-Left*: Example scene from the **Stack-B** task. *Center-Right* and *Right*: Tasks with unseen **distractors**.

6 Experiments

We evaluate our approach on several challenging long-horizon manipulation tasks in simulation (see sections D & F of the supplementary for details). Tasks involve the agent controlling a Sawyer manipulator equipped with a Robotiq gripper with a 5-dim. control (N_A= 5) to interact with a red and blue block on a tabletop. Observations are 64x64px RGB images from two cameras located on either side of the table, looking at the robot, and proprioceptive features. The latent representation is 128-dimensional (N_H=128) and unless otherwise noted we use a history of $H = 3$ observations and a rollout horizon of $N = 5$.

Task Setup: Fig.2 presents visualizations of a subset of tasks from our experiments. We consider three main tasks: 1) the **Lift** task requires the robot to lift an object above a certain threshold; *Lift-R* refers to lifting the red block and analagously for *Lift-B*. 2) the **Stack** task requires the robot to stack one object on top of the other; *Stack-R* refers to stacking the red block on top of the blue block and vice-versa for *Stack-B*. 3) Lastly, the **Match Positions** task involves moving both objects to a fixed target position. We also consider variants of these tasks with the addition of **visual distractors** and **stochasticity**. It is worth noting that all our tasks involve long-term dependencies and complicated contact dynamics making them particularly challenging for model-based approaches. We use shaped rewards for all tasks except the *Match Positions* task which has a mixed dense-sparse reward.

Baselines: We consider the following pixel-based baselines for our experiments: 1) **SVG(0)**: the model-free version of SVG. As our approach (termed Imagined Value Gradients – **IVG** from here on out) builds on SVG we expect to improve on SVG(0). 2) **MPO**: Maximum a Posteriori Policy Optimisation [10], a state-of-the-art model-free approach. To obtain an upper bound on performance we also include a version of MPO with access to the full system state (incl. objects). For the transfer experiments we also experiment with variants where we replace the value gradient based optimization. In particular we use **CEM**: the cross-entropy method [47] using the same model as IVG for transfer (latent rollouts). **PG**: replacing the value gradients with a likelihood ratio estimator (using 100 imagined rollouts), again using the same model as used for the *IVG* transfer. Additional details on the baselines are given in the appendix.

6.1 Learning from scratch

We first compare IVG and the baselines when learning the Lift-R and Stack-R manipulation tasks *from scratch*: Model-based IVG(5) learns the simpler *lift* task stably and performs on par with the baselines (Fig.3, left). On the harder stack task (Fig.3, right), IVG(5) learns significantly (about 2x) faster than both MPO and SVG(0), and also outperforms SVG(0) in terms of final performance. Compared to the informed MPO (State) baseline, IVG learns more slowly, but this difference is significantly reduced for stacking. Even when learning from pixels, the structure inherent in IVG via the latent space rollouts allows it to learn complex tasks faster than strong model-free baselines.

We also tested the ability of IVG to handle slightly more stochastic and partially observed environments. Fig.3 (right) presents results on learning the Stack-R task in environments with: 1) delayed proprioception (2 timestep lag) and 2) noisy observations where one of the blocks switches colors randomly every 3 frames. IVG(5) successfully learns in both cases (albeit more slowly). MPO also solved these tasks but required approximately $2x$ more episodes than IVG (color switch not shown).

6.2 Transferring learned models to related tasks

IVG learns a model of the environment which we may be able to transfer, and thus accelerate learning of related tasks. In the following, we evaluate this possibility. First, we train IVG from scratch

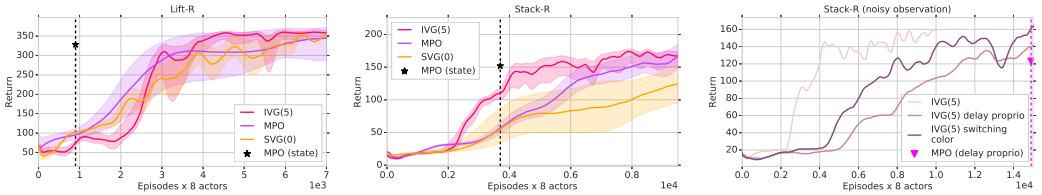


Figure 3: *Left*: IVG(5), with $N = 5$, performs similarly to the model-free baselines MPO and SVG(0) on the **lift** task. Black dotted line represents MPO (state) scoring higher than a reward threshold indicating success (300 for Lift, 140 Stack) for 50 episodes. *Center*: IVG(5) learns significantly faster than SVG(0) and MPO on the **stack** task. *Right*: IVG(5) learns well even in environments with noisy observations.

on one or multiple source task(s). Second, we copy the weights of the trained model (**encoder** f_{enc}, **transition** model f_{trans} and **decoder** f_{dec}). The policy, value and reward models are not transferred and are learned from scratch. As a model-free baseline we include MPO where we initialize all weights of the policy & value function except the last layer from an agent trained on a source task.

Multiple source tasks: A model trained on multiple tasks should transfer better due to the diversity in transitions observed. To test this hypothesis, we propose a version of IVG that is trained on multiple source tasks – learning a task agnostic model. We use the following source tasks: Reach-B, Move-B, Lift-B, Stack-B and Reach-R (most tasks involve the blue object but Reach-R gives the model some experience with the red block). Details on this setup are provided in the supplementary.

Transfer results: We present results on transferring IVG models to the following target tasks:

1) **Lift-R:** Fig.4 (left) shows the transfer performance of IVG(5) on the Lift-R task. With a model pre-learned on *Stack-B*, IVG(5) learns ∼2x faster than from scratch and ∼4x faster than MPO (irrespective of MPO's initialization). A model pre-trained on *Stack-R* accelerates IVG(5) further since the model has already observed many relevant transitions, achieving speed comparable to *MPO (state)*. Replacing the learned policy with direct optimization (*CEM*) or using a policy gradient yields sub-optimal behavior. These results highlight the benefits of a model when transferring to related tasks, even in cases where the model has only observed a subset of relevant transitions.

2) **Stack-R:** Results with transferred models for Stack-R are similar to those for Lift-R (Fig.4, center). But here, *CEM* fails to perform the stack (possibly caused by overly exploiting the model due to missing policy regularization), and using *PG* instead of a value gradient takes significantly longer to converge (15k trajectories) and performs worse, likely due to the noise in the likelihood ratio calculation – even though we already used 100 forward rollouts for the likelihood ratio calculation (1 for IVG), a 25 fold increase in computation. In addition, we also compared the performance when transferring a model trained on multiple source tasks (*Multitask*). This multi-task variant significantly accelerates learning speed; it is ∼1.5x faster than transferring from a single-task and about $\sim 3x$ faster than learning from scratch. As we will see from subsequent results, models trained on multiple tasks greatly accelerate transfer.

3) **Match Positions:** We tested model generalization on the *match positions* task, which differs significantly from the source tasks (Fig.4, right). All agents except multi-task IVG fail on this task; this includes transferring IVG from a single task (Stack-B) and the pixel-based MPO. This is likely due to the sparse structure of the reward (see supplementary). This shows how multi-task training enables the formation of robust and expressive latent spaces that transfers well to new tasks.

4) **Visual distractors:** Lastly, we analyzed the generalization of IVG when visual distractors in the form of a yellow cube or ball are added to the scene (Fig.5, (left & center)). Transfer from one or multiple source tasks remains effective. Transferring a multi-task model significantly outperforms all other methods, learning ∼3x faster than from scratch and ∼4x faster than MPO. Thus, even though CNNs are known to be sensitive to changes in visual inputs, jointly training a predictive model with the policy can still lead to robust representations that transfer quickly.

6.3 Ablation experiments

To validate our algorithm and model design choices we performed two sets of ablations.
Rollout horizon: To evaluate the effect of the rollout horizon N on the policy gradient (eq. (8)) we tested IVG across multiple settings of N=0,1,5,20 for learning *Stack-R* from scratch (Fig.5 (right)).

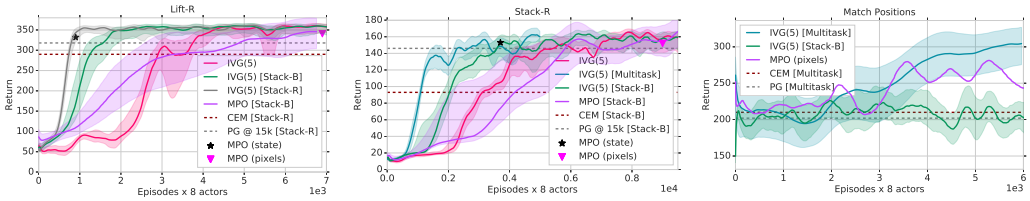


Figure 4: Transfer performance of IVG(5) on various target tasks. Task names inside square brackets indicate the source task. *Left*: Transfer results on Lift-R. Transferring from Stack-B leads to large improvements compared to baselines. *Center*: Transfer from multi-task IVG outperforms all baselines and single-task IVG. *Right*: Multi-task IVG successfully transfers on the Match Positions task while single-task IVG and other baselines fail.

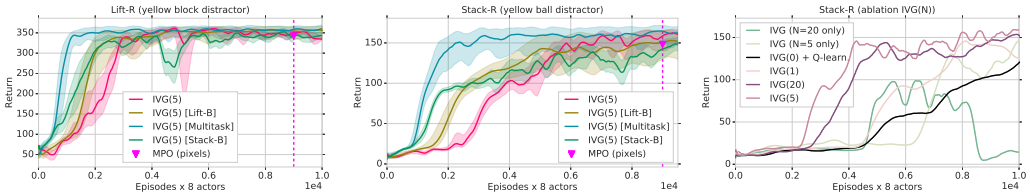


Figure 5: *Left & Center*: Transfer performance on the visual distractor task where a yellow block or ball are added to the Lift-R and Stack-R task respectively. transfer from multi-task IVG leads to a $\sim$3-4x speedup *Right*: Ablation tests with IVG, using horizon lengths of $N = 0, 1, 5, 20$ & using a single N-step horizon for computing the policy gradient (Eq. (8) instead of Eq. (9)). IVG(0) uses the model loss (with horizon 5) only as an auxiliary signal (and otherwise corresponds to SVG(0)).

IVG is robust to the choice of the rollout horizon and learns stably for all settings of N. However, the choice of N has a marked effect on the speed of learning. Increasing N speeds up learning up to a point after which no additional speedup is obtained (compare $N = 0, 1, 5$). Importantly, these results suggest that the benefit of the model in IVG is not just one of representation learning in partially observed domains but that using the model for policy optimziation is beneficial.

Averaging value gradients: To quantify the effect of averaging value gradients across multiple rollout horizons (c.f. Eq (9)), we ran experiments using a single imagined rollout of length N (Fig.5, right). For short horizons e.g. N=5, using a single rollout horizon leads to lower learning speed. On the other hand, learning completely fails for longer horizons N=20 (unlike with averaging) potentially due to cascading model errors on imagined rollouts, validating our averaging approach.

7 Conclusions

We presented an approach for model-based RL where an action-conditional latent space model is trained jointly with policy, value and reward functions that operate on the learned latent space. To achieve efficient policy optimization we introduced Imagined Value Gradients (IVG), an extension of SVG (using imagined rollouts and N-step horizon averaging). We demonstrated that IVG can learn complex, long-horizon manipulation tasks like lifting and stacking. We further demonstrated in several transfer experiments on related tasks that transferring a model learned via IVG can significantly improve data efficiency compared to off-policy baselines. Crucially, transferring with models trained on multiple tasks further accelerates learning, even succeeding on tasks where single-task transfer fails. We feel that our approach is a promising first step towards designing RL methods that combine learning of closed loop policies with the generalization capabilities that learned approximate models can provide – although extensions (such as handling egocentric cameras, increasing sample efficiency) are needed to make our approach a fully general purpose solution for real-world robotics tasks.

Acknowledgments

The authors would like to thank the entire Control Team and many others at DeepMind for numerous discussions on this work. Special thanks go to Tuomas Haarnoja and Raia Hadsell for reviewing an early version of this work and to Hannah Kirkwood for help in organizing the internship.

References

[1] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[2] OpenAI. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, et al. A general reinforcement learning algorithm that masters chess, shogi, & go through self-play. *Science*, 362(6419), 2018.

[4] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, et al. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II, 2019.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[6] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, et al. Model-based reinforcement learning for atari. *CoRR*, abs/1903.00374, 2019.

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[9] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *NeurIPS*, 2015.

[10] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[11] A. Abdolmaleki, J. T. Springenberg, J. Degrave, S. Bohez, Y. Tassa, D. Belov, et al. Relative entropy regularized policy iteration. *arXiv preprint arXiv:1812.02256*, 2018.

[12] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[13] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *ICRA*, 2019.

[14] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[15] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[16] F. Ebert, C. Finn, S. Dasari, A. Xie, A. X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, abs/1812.00568, 2018.

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

[18] D. Hafner, T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

[19] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *CoRR*, abs/1808.09105, 2018.

[20] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[21] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*. Elsevier, 1990.

[22] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *ICML*, 2016.

[23] T. Weber, S. Racanière, D. P. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, et al. Imagination-augmented agents for deep reinforcement learning. *CoRR*, abs/1707.06203, 2017.

[24] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[25] G. Wayne, C.-C. Hung, D. Amos, M. Mirza, A. Ahuja, A. Grabska-Barwinska, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

[26] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

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

[28] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, et al. Learning by playing-solving sparse reward tasks from scratch. In *ICML*, 2018.

[29] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From pixels to torques: Policy learning with deep dynamical models. *arXiv preprint arXiv:1502.02251*, 2015.

[30] N. Watters, L. Matthey, M. Bosnjak, et al. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *CoRR*, abs/1905.09275, 2019.

[31] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *NeurIPS*, 2018.

[32] C. Gelada, S. Kumar, J. Buckman, O. Nachum, and M. G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *ICML*, 2019.

[33] V. François-Lavet, Y. Bengio, D. Precup, and J. Pineau. Combined reinforcement learning via abstract representations. *arXiv preprint arXiv:1809.04506*, 2018.

[34] J. Oh, S. Singh, and H. Lee. Value prediction network. In *NeurIPS*, 2017.

[35] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, et al. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[36] A. Nagabandi, C. Finn, and S. Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. *CoRR*, abs/1812.07671, 2018.

[37] K. Hausman et al. Learning an embedding space for transferable robot skills. In *ICLR*, 2018.

[38] C. Florensa, Y. Duan, and P. Abbeel. Stochastic neural networks for hierarchical reinforcement learning. In *ICLR*, 2017.

[39] D. Tirumala, H. Noh, A. Galashov, L. Hasenclever, A. Ahuja, G. Wayne, et al. Exploiting hierarchy for learning and transfer in kl-regularized RL. *CoRR*, abs/1903.07438, 2019.

[40] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. In *NeurIPS 30*, 2017.

[41] A. Galashov, S. Jayakumar, L. Hasenclever, D. Tirumala, J. Schwarz, G. Desjardins, W. M. Czarnecki, Y. W. Teh, et al. Information asymmetry in KL-regularized RL. In *ICLR*, 2019.

[42] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[43] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel. Model-based reinforcement learning via meta-policy optimization. In *CoRL*, 2018.

[44] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2013.

[45] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[46] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[47] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer, 2013.

We provide additional details on some of the model components as well as full details regarding our experimental setup.

A Additional details on Value gradient derivation

Given a model, we optimize a parametric policy $\pi_\theta(\mathbf{a}|\mathbf{h})$ by maximizing the N-step surrogate value function which is a recursive composition of the policy, the transition, reward and value function:

$$\tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{\mathbf{a}^k \sim \pi}\left[\gamma^N \hat{V}^\pi(\mathbf{h}^{t+N};\phi) + \sum_{k=t}^{t+N-1} \gamma^{k-t}\hat{r}(\mathbf{h}^k,\mathbf{a}^k)\Big|\mathbf{h}^{k+1} = f_{\text{trans}}(\mathbf{h}^k,\mathbf{a}^k)\right], \quad (1)$$

This N-step value can be computed by performing an "imagined" rollout in the latent state-space using our model (see Figure (1) in the main text). It can be maximized by gradient ascent, exploiting the so called "value gradient" $\nabla_\theta \tilde{V}_N(\mathbf{h}^t)$ [1]; which can often be computed recursively, taking advantage of the reparameterization trick [2, 3] for sampling from π_θ we can recursively define a sample estimate of this gradient. We start by defining the deterministic function $\pi_\theta(\mathbf{h}^t,\epsilon)$ that transforms a sample ϵ from a canonical noise distribution $p(\epsilon)$ into a sample from $\pi_\theta(\mathbf{a}|\mathbf{h})$. In the following we will consider Gaussian policy distributions, i.e. $\pi_\theta(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{h}), \sigma^2_\theta(\mathbf{h}))$, for which the reparameterization is given as $\pi_\theta(\mathbf{h}^t,\epsilon) = \mu_\theta(\mathbf{h}) + \epsilon\sigma_\theta(\mathbf{h})$, with $p(\epsilon) = \mathcal{N}(\mathbf{0},\mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix. Using these definitions we can define the gradient $\nabla_\theta \tilde{V}_N(\mathbf{h}^t)$ for any state as

$$\nabla_\theta \tilde{V}_N(\mathbf{h}^t) = \mathbb{E}_{p(\epsilon)}\left[\nabla_\theta \hat{r}\big(\mathbf{h}^t,\pi_\theta(\mathbf{h}^t,\epsilon)\big) + \gamma \nabla_{\mathbf{h}'}\tilde{V}_{N-1}(\mathbf{h}')\nabla_\theta f_{\text{trans}}\big(\mathbf{h}^t,\pi_\theta(\mathbf{h}^t,\epsilon)\big) + \gamma\nabla_\theta \tilde{V}_{N-1}(\mathbf{h}')\right] \quad (2)$$

where $\mathbf{h}' := \mathbf{h}^{t+1} = f_{\text{trans}}(\mathbf{h}^t,\pi_\theta(\mathbf{h}^t,\epsilon))$ and we dropped dependencies of all functions on ϕ for brevity. The partial value gradient $\nabla_{\mathbf{h}'}\tilde{V}_{N-1}(\mathbf{h}')$ wrt. a state $\mathbf{h}'$ is defined recursively as

$$\begin{aligned}\nabla_{\mathbf{h}^k}\tilde{V}_N(\mathbf{h}^k) = \mathbb{E}_{p(\epsilon)}\Big[&\nabla_{\mathbf{h}^k}\hat{r}\big(\mathbf{h}^k,\mathbf{a}^k\big) + \nabla_{\mathbf{a}^k}\hat{r}\big(\mathbf{h}^k,\mathbf{a}^k\big)\nabla_{\mathbf{h}^k}\pi_\theta\big(\mathbf{h}^k,\epsilon\big) + \\ &\gamma\nabla_{\mathbf{h}^{k+1}}\tilde{V}_{N-1}\big(\mathbf{h}^{k+1}\big)\nabla_{\mathbf{h}^k}f_{\text{trans}}\big(\mathbf{h}^k,\mathbf{a}^k\big) + \\ &\gamma\nabla_{\mathbf{h}^{k+1}}\tilde{V}_{N-1}\big(\mathbf{h}^{k+1}\big)\nabla_{\mathbf{a}^k}f_{\text{trans}}\big(\mathbf{h}^k,\mathbf{a}^k\big)\nabla_{\mathbf{h}^k}\pi_\theta\big(\mathbf{h}^k,\epsilon\big)\ \Big|\ \mathbf{a}^k = \pi_\theta\big(\mathbf{h}^k,\epsilon\big)\Big]\end{aligned} \quad (3)$$

where the case $\nabla_\theta \tilde{V}_1(\mathbf{h}^{t+N-1})$ is established by assuming that the policy does not change in any step after N; i.e. bootstrapping with $\nabla_{\mathbf{h}^k}\tilde{V}_0(\mathbf{h}^k) = \nabla_{\mathbf{h}^k}\hat{V}^\pi(\mathbf{h}^k;\phi)$. We note that, to calculate these gradients, only an initial state $\mathbf{h}^t$ (encoded from a history of observations $\mathbf{o}^{1:t} \sim \mathcal{B}$) is required in addition to the learned model. Our derivation here is thus analogous to the N-step stochastic value gradient definition from [1] but replacing observed states with imagined latent states – and assuming a deterministic transition model.

B Additional details on Regularized Policy Optimization

Given the definition of the value gradient in the previous section we can make a few interesting observations relevant to its use in practice.

First, we can realize that, in principle, the single-step gradient estimate $\nabla_\theta \tilde{V}_1(\mathbf{h})$ (and hence a model trained by minimizing $\mathcal{L}^1$ is sufficient for performing policy optimization. However, in this case we would obtain a biased value gradient after one gradient step in the direction of $\nabla_\theta \tilde{V}_1(\mathbf{h})$ – since at that point $\hat{V}^\pi(\mathbf{h}) \neq \tilde{V}^{\pi_\theta}(\mathbf{h})$ – which only becomes unbiased again once the dynamic programming updates from (7) have converged. To counteract this bias we could consider using the N-step gradient $\nabla_\theta \tilde{V}_N(\mathbf{h})$ for large N. Such an estimate is not affected by the above described bias for the first $N-1$ steps (since the equivalence of π and π_θ is only assumed for steps after time N). As a result, it facilitates faster learning (as also demonstrated in our experiments). A downside of this

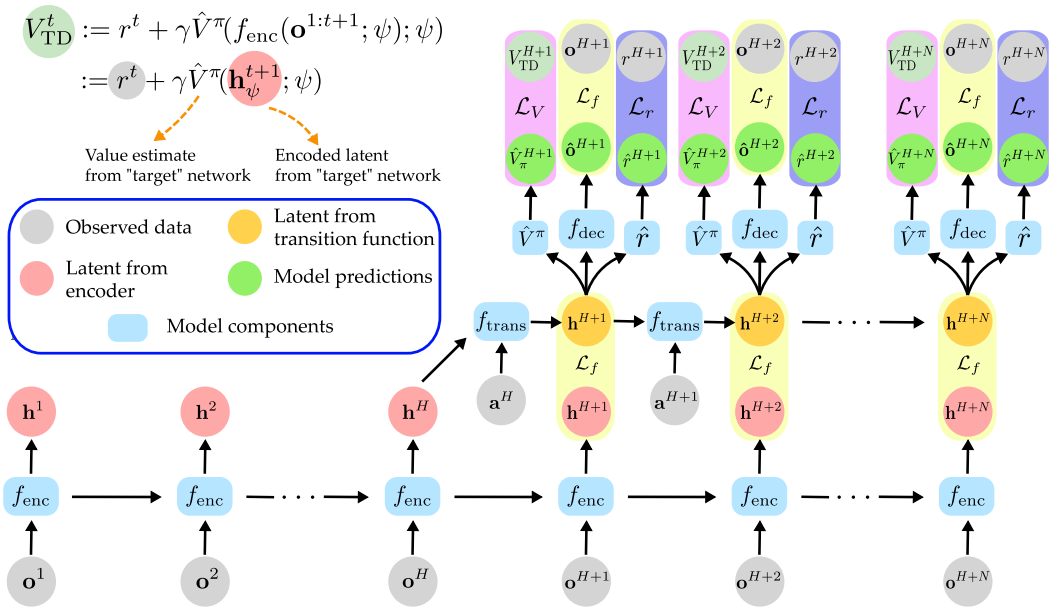


Figure 1: Schematic showing the rollout from on a trajectory sampled from the replay buffer $(\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N}) \sim \mathcal{B}$ through the model (blue rectangles), predicting latent states (red & orange circles for encoder and transition predictions respectively) and their corresponding reconstructed observations, value and reward predictions (green circles). First, the encoder f_{enc} encodes the observations $\mathbf{o}^{1:H+N}$ to latents $\mathbf{h}^{1:H+N}$. The open-loop rollout begins after a history of H observations have been encoded to generate the latent $\mathbf{h}^{H}$. From this latent, the rollout is computed through the transition model f_{trans} using the true actions $\mathbf{a}^{H:H+N-1}$, generating the latents $\mathbf{h}^{H+1:H+N}$ (orange circles; note that these are different from the latents generated by the encoder). The transition model latents are passed through the decoder f_{dec}, value $\hat{V}^{\pi}$ and reward $\hat{r}$ estimators to generate (expected) reconstructed observations $\hat{\mathbf{o}}^{H+1:H+N}$, (expected) values $\hat{V}_{\pi}^{H+1:H+N}$ and (expected) rewards $\hat{r}^{H+1:H+N}$ which are used to compute losses for training the model (losses are highlighted with the rectangular color patches with the labels $\mathcal{L}_{(.)}$). Of special mention is the loss for training the value estimator; this uses V-trace and Temporal Difference (TD) style value targets based on a "target" network. The targets are generated by the "target" value estimator $\hat{V}^{\pi}(.\,;\psi)$ which takes in latents encoded by the "target" encoder $f_{\text{enc}}(.\,;\psi)$ along with the observed rewards r^{t}. Best viewed in color.

approach is that it can be more heavily affected by modelling errors; i.e. a latent state-space model predicting rewards N-steps into the future is harder to learn than a 1-step model. As a compromise, trading-off bias with modelling errors, we found that using a simple average gradient estimate – over N horizons – worked well in practice $\nabla_{\theta}\bar{V}_{N}(\mathbf{h}) = \frac{1}{N}\sum_{k}^{N}\nabla_{\theta}\tilde{V}_{k}(\mathbf{h})$. This averaging linearly down weights the contributions from states further along the trajectory – the first state appears N times in the sum, N-1 times for the second state and so on; as opposed to a discount based weighting that decays slowly this can drastically reduce the effect of model errors later in the sequence. We note that, in principle, we could also use weighting terms based on the variance of different horizon estimates, but opted for an average for simplicity here.

Second, even with the averaged model-gradient from above, gradient based optimization is prone to exploiting modelling errors (in both the transition dynamics and reward/value estimates), yielding overly optimistic policies. This is a well known problem in model based RL; see e.g. [4] for a recent discussion. To counteract such effects it is hence desirable to further regularize the policy optimization step. Similar to many existing policy optimization methods [5, 6, 7] we adopt a relative-entropy (KL) regularization scheme. We augment the estimated reward with a sample based likelihood ratio term (a sample based estimate of the KL)

$$\hat{r}_{\text{KL}}(\mathbf{h}, \mathbf{a}, \pi_{\theta}) = \hat{r}(\mathbf{h}, \mathbf{a}) + \lambda \log \frac{\pi_{\theta}(\mathbf{a}|\mathbf{h})}{p(\mathbf{a}|\mathbf{h})}, \tag{4}$$

Algorithm 1 Imagined Value Gradients in Latent Spaces

Given: Empty experience dataset $\mathcal{B}$, burn-in H, rollout length N, episode length T
Each actor do:
  while True **do**
    Fetch policy / model parameters (θ, ϕ) from learner; Initialize empty trajectory $\tau := \emptyset$
    for t = 0 to T **do**
      Apply control $\pi_\theta(f_{enc}(\mathbf{o}^{t-H:t}; \phi), \epsilon),\ \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ # Encode history, take action
      Observe $r, \mathbf{o}'$; Insert $(\mathbf{o}, \mathbf{a}, r, \mathbf{o}')$ into τ
    end for
    Save τ in the dataset $\mathcal{B}$
  end while
Learner do:
  while True **do**
    Sample sub-trajectory of length $H + N$ from buffer: $(\mathbf{o}^{1:H+N}, \mathbf{a}^{1:H+N}, r^{1:H+N}) \sim \mathcal{B}$
    Compute $\nabla_\phi \mathcal{L}^{\mathcal{N}}$, the gradient of Eq. (3) w.r.t model, reward and value parameters ϕ
    Compute the policy gradient $\nabla_\theta \mathbb{E}_{p_\pi}\left[\bar{V}_N^{\text{KL}}(\mathbf{h})\right]$ (Eq. (9)) w.r.t policy parameters θ
    Grad ascent/descent: $\theta \leftarrow \text{Step}_{\text{adam}}(\theta, \nabla_\theta \mathbb{E}_{p_\pi}\left[\bar{V}_N^{\text{KL}}(\mathbf{h})\right]);\ \phi \leftarrow \text{Step}_{\text{adam}}(\phi, -\nabla_\phi \mathcal{L}^N)$
  end while

where $p(\mathbf{a}|\mathbf{h})$ is a the prior action probability (we use $p(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ throughout) and λ is a multiplier trading-off reward and regularization. replacing $\hat{r}$ in Equation (2) with $\hat{r}_{\text{KL}}$ – noting that $\hat{r}_{\text{KL}}$ is differentiable wrt. the policy parameters θ – results in the regularized value gradient

$$\begin{aligned}\nabla_\theta \tilde{V}_N^{\text{KL}}(\mathbf{h}) = \mathbb{E}_{p(\epsilon)}\Big[&\nabla_\theta \hat{r}_{\text{KL}}\big(\mathbf{h}, \pi_\theta(\mathbf{h}, \epsilon), \pi_\theta\big) + \\ &\gamma \nabla_{\mathbf{h}'} \tilde{V}_{N-1}^{\text{KL}}(\mathbf{h}') \nabla_\theta f_{\text{trans}}\big(\mathbf{h}, \pi_\theta(\mathbf{h}, \epsilon)\big) + \\ &\gamma \nabla_\theta \tilde{V}_{N-1}^{\text{KL}}(\mathbf{h}')\Big],\end{aligned} \tag{5}$$

where $\nabla_{\mathbf{h}'} \tilde{V}_{N-1}^{\text{KL}}(\mathbf{h}')$ is, analogously, given by inserting $\hat{r}_{\text{KL}}$ into Equation (3). To ensure that the bootstrap value for $\tilde{V}_0^{\text{KL}}(\mathbf{h})$ is compatible with this regularized reward we additionally change the loss for the value function (the Bellman error); by, again, replacing $\hat{r}$ with $\hat{r}_{\text{KL}}$ in Equation (7). The total derivative estimate we use in practice is then given as

$$\nabla_\theta \mathbb{E}_{\bar{p}}\left[\bar{V}_N^{\text{KL}}(\mathbf{h}^t)\right] \approx \mathbb{E}_{\mathbf{o}^{1:t} \sim \mathcal{B}}\Big[\frac{1}{N} \sum_{k=1}^{N} \nabla_\theta \tilde{V}_k^{\text{KL}}(f_{\text{enc}}(\mathbf{o}^{1:t}))\Big], \tag{6}$$

where we use batches of samples from the replay to optimize the policy on all visited states. That is we perform stochastic gradient ascent combining the gradient from Equation (6) with any optimization method (we use Adam [8]). The full optimization procedure is also described in Algorithm. 1.

C Details for the Value Learning step

As described in the main paper value-loss $\mathcal{L}_V$ involves the calculation of a (squared) Bellman error, which is given by

$$\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1}) = \frac{\pi(\mathbf{a}^t|\mathbf{h}^t)}{\mu(\mathbf{a}^t|\mathbf{h}^t)} \left(r^t + \gamma \hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) - \hat{V}^\pi(\mathbf{h}^t; \phi)\right)^2, \tag{7}$$

where the next state value $\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi)$ is calculated via a "target network", whose parameters ψ are periodically copied from ϕ, to stabilize training (see e.g. [9] for a discussion). In practice we use v-trace [10] to calculate a better target value. That is we set $\hat{V}^\pi(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi) \hat{=} \hat{V}^\pi_{\text{trace}}(f_{\text{enc}}(\mathbf{o}^{1:t+1}; \psi); \psi)$, where the v-trace target is given as:

$$\hat{V}^\pi_{\text{trace}}(\mathbf{h}^t; \psi) = \hat{V}^\pi(\mathbf{h}^t; \psi) + \sum_{i=t}^{t+N-1} \gamma^{i-t} \left(\Pi_{j=t}^{i-1} \rho_i\right) \delta_i^V \tag{8}$$

with $\delta_i^V = \rho_i(r_i + \gamma\hat{V}^\pi(\mathbf{h}_{i+1};\psi) - \hat{V}^\pi(\mathbf{h}^t;\psi))$ being the temporal difference error multiplied by importance weight $\rho_i = \min\left(\rho_{\text{clip}}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}\right)$ with $\mu(a|s)$ denoting the behaviour policy and we set $\rho_{\text{clip}} = 1$. We refer to Espeholt et al. [10] for additional details regarding v-trace.

D Details for the Experimental Setup

We used the Mujoco Simulator[1] for simulating the Sawyer robot setup. The robot is equipped with a two-finger Robotiq gripper. We ran the simulation with a numerical time step of 10 milliseconds, integrating 5 steps, to get a control interval of 50 milliseconds for the agent. In this way we can resolve all important properties of the robot arm and the object interactions in simulation. All the objects used were based on wooden toy blocks and balls. For the majority of our experiments we used two cubic blocks with side lengths of 5 cm, colored red and blue. For the distractor experiments, we used a yellow cubic block of size 6 cm and a yellow ball of diameter 4 cm. We used a table with sides of 60 cm x 30 cm in length as the workspace of the robot in all our experiments. Objects were spawned randomly on the table surface. The robot hand is initialized randomly above the table-top with a height offset of up to 20 cm above the table (minimum 10 cm) and the fingers in an open configuration. All experiments run on episodes with 200 steps length (which gives a total simulated real time of 10 seconds per episode).

The sawyer robot is controlled via a 5D position controller based on an inverse kinematics model. That is, that agent outputs are 5D velocity commands – 3 for the robot end effector position, one for the gripper and one for rotating the gripper to change its orientation. The action space for all our tasks, therefore, is 5 dimensional.

Entry	Dimensions	Unit
arm joint pos	7	rad
arm joint vel	7	rad / s
finger joint pos	1	rad
finger joint vel	1	rad / s
finger grasp	1	Binary

Entry	Dimensions	Unit
camera 1 (front-left)	3 x 64 x 64	RGB
camera 2 (front-right)	3 x 64 x 64	RGB

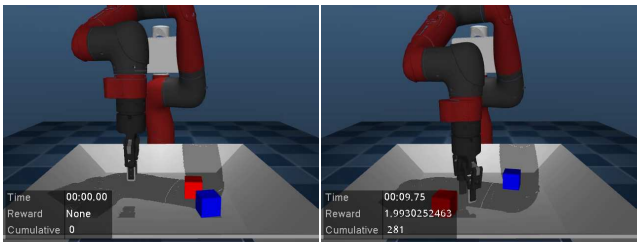
Table 1: *Left*: Proprioceptive observations used in all simulation experiments. *Right*: Image observations used in all simulation experiments (except the state based ones).

Entry	Dimensions	Unit
object i pose	7	m, au
object i velocity	6	m/s, dq/dt
object i relative pos	3	m

Table 2: Object feature observations, used in the state based MPO experiment. Note that these are not used for any vision based experiment. The pose of the objects is represented as world coordinate position and quaternions. In the table m denotes meters, q refers to a quaternion which is in arbitrary units (au). i denotes the id of the object; features from all objects are used as input.

Table 1 (left) shows the list of proprioception observations we use for all our experiments. These observations are concatenated to produce a 17 dimensional vector which is used as input to our model. In addition to proprioception, we use RGB images from two cameras located to the left and right of the table (in the front of the table, pointing towards the robot) as visual observations (see table 1, right). These two images (3x64x64 each) are concatenated along the channel dimensions to generate a 6x64x64 input visual observation to our model. It is worth noting that the availability of two camera views helps disambiguate most occlusions. We will explore switching to a single central view in future experiments; this significantly increases occlusions and makes the tasks strongly partially observable. For the baseline state based MPO experiment, we used features of the objects (see table 2) in addition to proprioceptive features as inputs to the policy.

[1] MuJoCo: see www.mujoco.org



Push (initial) Push (target)

Figure 2: Example scenes from the match positions task. On the left is an initial image showing the blocks and the arm initialized to random starting positions. On the right, we show an image where the blocks are in their respective target positions (the blocks are always required to go to this target configuration in the task).

Fig.8 shows example images from a learned policy on the IVG task; both camera views used for training are shown.

D.1 Tasks and Rewards

We used shaped rewards for specifying all our tasks as we wanted to measure transfer efficiency rather than the capability to handle sparse reward settings. In principle, the multi-task version of IVG can be extended to the sparse reward setting easily, in a manner similar to SAC [11]. Below, we discuss the reward setup for the three major tasks considered in the paper, the **Lift**, **Stack** and **Match Positions** task. The addition of a *visual distractor* does not change the reward setup for a given task.

Lift: In the lift task, the agent has to pick-up one of either the red (Lift-R) or blue (Lift-B) objects on the table and lift it above a certain height. We introduce additional shaping to the task through auxiliary rewards that encourage reaching the target object, grasping it and lifting it once grasped. These are specified in turn as:

- *REACH(O)*: $tol(d(TCP, O), 0.02, 0.15)$:
  Minimize the distance of the TCP to the target cube.
- *GRASP*:
  Activate grasp sensor of gripper ("inward grasp signal" of Robotiq gripper)
- *HEIGHT(O, x)*: $lin(O, x, 0.10)$
  Increase z coordinate of an object more than $x = 0.03m$ relative to the table.

Where the $d(x, y)$ is the Euclidean distance between a pair of 3D points, and the tolerance and linear reward functions terms are defined as:

$$tol(v, \epsilon, r) = \begin{cases} 1 & \text{iff } |v| < \epsilon \\ 1 - tanh^2(\frac{atanh(\sqrt{0.95})}{r}|v|) & \text{else,} \end{cases} \tag{9}$$

$$lin(v, \epsilon_{min}, \epsilon_{max}) = \begin{cases} 0 & \text{iff } v < \epsilon_{min} \\ 1 & \text{iff } v > \epsilon_{max} \\ \frac{v - \epsilon_{min}}{\epsilon_{max} - \epsilon_{min}} & \text{else.} \end{cases} \tag{10}$$

The final reward is a weighted sum of all these sub-rewards:

$$LIFT(O) = REACH(O) + 0.5 * (GRASP + HEIGHT(O, 0.03)),$$

which, overall cannot exceed a value of two.

Stack: Similar to the lift task, there are two variants of the stack task: 1) Stack-R, where the agent has to stack the red block on the blue block and 2) Stack-B, where the agent does the opposite. We again introduce shaping by first encouraging the agent to lift the object – the lift reward is a part of the reward for the stack task. Additionally, once the object has been lifted we encourage the agent

to move towards the target, align it with the target block and release the grasped object. The total reward is:

$$STACK(O1, O2) = \begin{cases} LIFT(O1) & \text{iff } HEIGHT(O1, 0.03) \leq 0.8 \\ STACKED(O1, O2) & \text{else.} \end{cases} \tag{11}$$

where:

$$STACKED(O1, O2) = ABOVE(O1, O2) * NOTGRASP,$$

where $NOTGRASP$ is detemined by the grasp sensor and:

$$ABOVE(O1, O2) = tol(d(O1, O2), 0.02, 0.15) * HEIGHT(O1, 0.03)$$

Match Positions: In this task, the agent has to move both the red and blue blocks to a fixed target position (see Fig.2). As this task involves moving both objects it is a nice setting for testing the generalization of our learned models. Additionally, the reward is not shaped to encourage motion towards an object; there is no change in the reward unless one of the objects is moved. We specify the reward as:

$$MP(O1, O2) = tol(d(O1, t1), 0.02, 0.15) + tol(d(O2, t2), 0.02, 0.15) \tag{12}$$

where $t1, t2$ denote the target 3D positions of the red and blue block respectively.

D.2 Partially observable environments

For the experiments using more partially observable environments we considered two settings. In the first instance we added a delay to the proprioceptive features in the Stack-R task (see IVG(5) (delayed proprio) in Figure 3 in the main paper). This results in an environment where the RNN has to perform integration to estimate the robot arm position (and its velocities).

In the second experiment we created a variant of the Stack-R task in which the red block (that needs to be lifted and placed on top of the blue block) changes color (switching from blue to red at random every 2 frames).

D.3 Multi-task setup: Tasks and Rewards

In the multi-task setup we introduce several auxiliary tasks that are solved in addition to a main extrinsic task. We consider the following tasks and rewards in all our multi-task experiments:

- $REACH(O_{\text{Red}})$
- $REACH(O_{\text{Blue}})$
- $LIFT(O_{\text{Blue}})$
- $STACK(O_{\text{Blue}})$
- $MOVE(O_{\text{Blue}})$

where the move reward is given by $MOVE(O) = tol(d(vel_O, v), 3, 0)$ where vel_O is the object's velocity. To train in a multi-task setup we use a task-conditioned policy, value- and reward-function, we refer to the section below for details. The learned model (i.e. $f_{\text{enc}}, f_{\text{trans}}, f_{\text{dec}}$) on the other hand is not conditioned on the task – it hence has to learn consistent dynamics across tasks. The actors generate data by selecting one task per episode at random.

We note that even though the multi-task largely consists of the same rewards as for individual experiments the data distribution is very different as the model is trained on episodes from all tasks. As can be seen, in the experiments in main paper, this results in significant improvements in the transfer learning experiments.

E Details on the Model & Policy

We present some details on the architecture of the model components and policy network below:

The **encoder** f_{enc} uses a recurrent, deterministic, convolutional neural network (CNN) to encode the observations $\mathbf{o}^t$ to a low-dimensional latent state representation $\mathbf{h}^t$ (see Fig.3). Our observation is a pair of RGB images (3x64x64 each), concatenated along the first dimension, and a proprioception vector. The images are passed through a CNN with an initial convolutional layer followed by three residual blocks with strided convolutions [12] and average pooling to generate a vector of outputs. In parallel, the proprioception input is passed through a 2-layer multilayer perceptron (MLP) to generate a feature vector. These are concatenated and passed through a 3-layer MLP and an LSTM which outputs the latent state $\mathbf{h}^t$. As an initial pre-processing step, we normalized all our images to be between 0-1 and proprioception to -1 to 1.

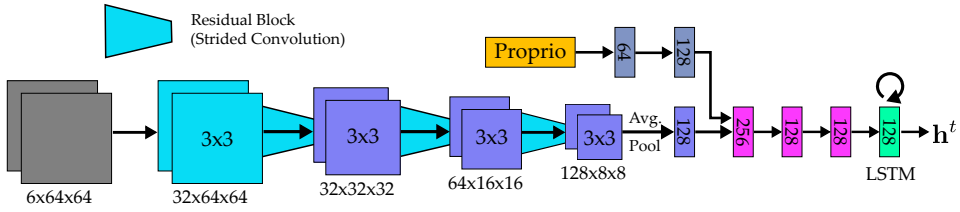


Figure 3: Network architecture of the encoder. The encoder takes in a pair of 64x64 RGB images concatenated along the channels axis and the proprioception observation and returns a 128-dimensional latent state vector ($\mathbf{h}$) as output. It is implemented as a recurrent residual CNN with a final LSTM layer that integrates information across time.

The **transition** model f_{trans} is deterministic, taking a latent state $\mathbf{h}^t$ and action $\mathbf{a}^t$ to predict the next latent state $\mathbf{h}^{t+1}$ (see Fig.4). Both the inputs are first passed through 2-layer MLPs. The outputs of these MLPs are concatenated and passed through another 2-layer MLP which predicts the change in latent state $\delta\mathbf{h}$. To ensure that the transition model outputs are well conditioned for long rollouts, we pass this delta change through a *tanh* layer to normalize the result to -1 to 1. This is further scaled by a linear transform and added to the input state $\mathbf{h}^t$ to generate the prediction $\mathbf{h}^{t+1}$. In practice, we saw a significant improvement in performance when predicting the change in state as opposed to directly predicting the next state.

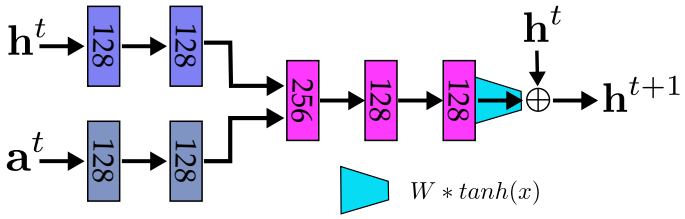


Figure 4: Network architecture of the transition model. The transition model takes a state ($\mathbf{h}$) and action ($\mathbf{a}$) as input and returns a prediction of the next state ($\mathbf{h}'$). It is implemented as an MLP that predicts a delta change to the state ($\delta\mathbf{s}$) which is added to the input state to predict the output.

The **decoder** f_{dec} predicts the (expected) input observation $\mathbf{o}^t$ from the latent state $\mathbf{h}^t$ (see Fig.5). We have two parts to the decoder: 1) To reconstruct the proprioception input, we pass the latent state through a 2-layer MLP. 2) For reconstructing the images, we first use a linear layer to transform the latent state to a 2048 dimensional vector which is reshaped into a 64x8x8 feature tensor. This feature tensor is passed through three upsampling layers, each using a bilinear additive upsampling layer [13] followed by a convolution; the output is at the same resolution as the input images. Finally, the output features are passed through a 1x1 convolution layer to get the correct number of channels and a sigmoid layer to ensure that the outputs are normalized. We also experimented with using a de-convolutional architecture for the image upsampling but found that it reduced the reconstruction quality.

The **value** $\hat{V}^\pi$ and **reward** $\hat{r}$ modules predict the (expected) value $\hat{V}^t_\pi := \hat{V}^\pi(\mathbf{h}^t; \phi)$ and reward $\hat{r}^t$ from a given state $\mathbf{h}^t$ (see Fig.6). Both these modules are implemented as 3-layer MLPs, with a

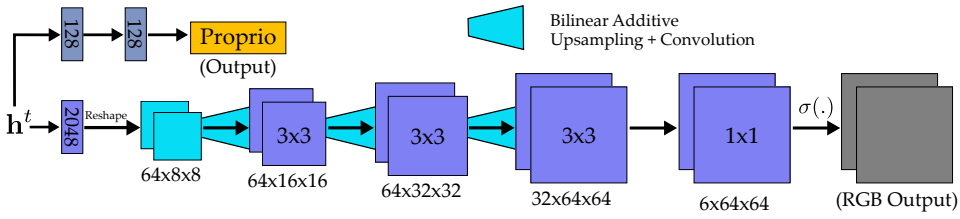


Figure 5: Network architecture of the decoder. The decoder takes a state ($\mathbf{h}$) as input and returns a reconstruction of the corresponding RGB images and proprioception. We use an MLP to predict the proprioception output and a mix of bilinear upsampling and convolutional layers to generate the RGB reconstructions (which are normalized to 0-1 via a sigmoid).

layer norm [14] after the output of first layer.

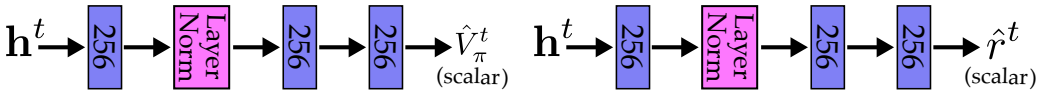


Figure 6: *Left*: Network architecture of the value model $\hat{V}^\pi$. The network takes the latent state $\mathbf{h}^t$ as input and uses a three fully-connected layers to predict the expected value $\hat{V}_\pi^t := \hat{V}^\pi(\mathbf{h}^t; \phi)$ (scalar). *Right*: The reward model uses the same architecture as the value model but predicts the immediate reward $\hat{r}^t$ from the state $\mathbf{h}^t$.

Lastly, the **policy** $\pi_\theta(\mathbf{a}|\mathbf{h})$ network predicts a distribution over actions $\mathbf{a}^t$ from the corresponding latent state $\mathbf{h}^t$. As mentioned earlier, we consider Gaussian policy distributions i.e. $\pi_\theta(\mathbf{a}|\mathbf{h}) = \mathcal{N}(\mu_\theta(\mathbf{h}), \sigma_\theta^2(\mathbf{h}))$, from which we can sample through the reparameterization trick as $\pi_\theta(\mathbf{h}^t, \epsilon) = \mu_\theta(\mathbf{h}^t) + \epsilon\sigma_\theta(\mathbf{h}^t)$, with $p(\epsilon) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ denotes the identity matrix. We implement the policy network as a 3-layer MLP similar to the **value** and **reward** modules. Unlike those, the policy outputs the mean μ_θ and log-standard deviation $log(\sigma_\theta)$ from which we can sample an action using the reparameterization shown above.

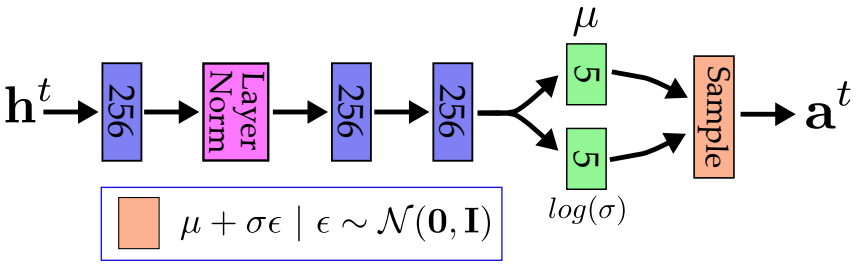


Figure 7: Network architecture of the policy $\pi_\theta(\mathbf{a}|\mathbf{h})$. The policy takes as input the state $\mathbf{h}^t$ and predicts the mean μ and log-variance $log(\sigma)$ of a Gaussian distribution over actions. We use the reparameterization trick to sample from this distribution by sampling $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The output is a sampled action $\mathbf{a}^t$.

E.1 Multi-task learning

We introduce a few additional changes to the network architecture of a few model components and the policy in the multi-task learning setup. In this setting, each task has a unique **task ID** id associated with it – this is represented as a 1-hot vector of length M (M is the number of tasks). This task ID is fed as an additional input to both the **policy** ($\pi_\theta(\mathbf{a}|\mathbf{h}, \text{id})$) and **value** modules ($\hat{V}^{\pi,\text{id}}$), thereby conditioning their predictions based on the task that is currently being considered. On the other hand, the **reward** predictor now predicts the rewards $\hat{r}^t$ for all these tasks; its output is now M-dimensional as opposed to a scalar from before. This further encourages the latent state to capture features relevant to all the learned tasks, leading to better generalization performance as witnessed in our experiments. Lastly, the architectures of the encoder f_{enc}, decoder f_{dec} and transition model

f_{trans} are unchanged; these components are task agnostic and can integrate & transfer knowledge across tasks.

F Details on training and transfer setup

We implemented all our models in Python using the Tensorflow neural network package. Below, we present some details on the loss functions used for training and the hyper-parameter settings.

F.1 Model Loss

We defined the per example model loss as $\mathcal{L}_e = \mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) + \alpha\mathcal{L}_r(\mathbf{h}^t, \mathbf{a}^t, r^t) + \beta\mathcal{L}_V(\mathbf{h}^t, \mathbf{a}^t, r^t, \mathbf{o}^{1:t+1})$. α and β are coefficients that determine the relative contribution of the loss components. As explained in the main text, we use a squared error term for the reward loss $\mathcal{L}_r$ and a squared error to a V-trace target for the value loss $\mathcal{L}_V$. The per example transition model loss is given as

$$\mathcal{L}_f(\mathbf{h}^t, \mathbf{o}^{1:t}) = \|f_{\text{dec}}(\mathbf{h}^t; \phi) - \mathbf{o}^t\|_2^2 + \zeta\|f_{\text{enc}}(\mathbf{o}^{1:t}; \phi) - \mathbf{h}^t\|_2^2, \tag{13}$$

where the first term measures the error between the observations $\mathbf{o}$ and reconstructions from the open-loop latent state predictions ($\mathbf{h}^{t>H}$), the second term enforces consistency between the latent states predicted by the encoder f_{enc} and the transition model f_{trans} and ζ is a coefficient that determines the relative contribution of the two loss terms. The reconstruction loss is split into two parts (weighted equally), an image reconstruction loss and a proprioception reconstruction loss. We use a squared error term for the proprioception loss and a binary cross entropy loss term for image reconstruction; in practice we found this to result in better image reconstructions than a squared error term.

F.2 Hyper-parameters

We used ADAM [8] with default settings and a fixed learning rate of 5e-5 for all our experiments. We used the ELU [15] non-linearity as the activation function in all our networks. We initialized the final layers of our policy to predict values close to zero at the start of training; we found that this improved stability, especially in the early stages of learning. We used a latent state dimension of $N_S = 128$ for all our experiments ($|\mathbf{h}| = 128$). We found this to be low-dimensional enough to be used for fast RL while still allowing room for expressivity.

We found that setting $\alpha = \beta = \zeta = 1.0$ gave the good results and kept this setting throughout all experiments. For the policy optimization, we set the weight of the KL regularizer to $\lambda = 0.01$ based on a hyper-parameter sweep.

We used a batch size of 32 (two learners each with a batch size of 16) to train our model and policy in all our experiments; we initially experimented with larger batch sizes of 128 but found that lowering the batch size made learning more stable. We fixed the history length $H = 3$ and experimented with different rollout lengths $N = 1, 5, 20$; as shown in our experiments $N = 5$ performed best, we use that as the default. We ran experiments for a fixed number of episodes (per actor).

F.3 Actor data generation

We used 8 asynchronous actors for data generation in all our experiments. At the start of each episode, the actor retrieves the most recent model and policy parameters. It then executes this policy a fixed time horizon of $T = 10$ seconds (episode lasts 10 seconds). The resulting trajectory is split up into smaller sub-trajectories of length $H + N$, the length needed for learning, and added to a central replay buffer which collects experience from all actors. We used a buffer containing up to $100,000$ sequences (randomly deleting old sequences when full) for all our experiments. Both our learners sample from this replay buffer, compute the gradients for the model components and policy and perform synchronized updates to the parameters.

For the multi-task experiments, at the start of each episode, the actor chooses a task to execute at random out of the M available tasks. This task is executed for the full length of the episode ($T = 10$ seconds). Random sampling of tasks can help generate diverse trajectories for training, facilitating learning of an expressive latent representation.

F.4 Baseline parameters

For the baseline experiments that used pixel observations we constructed policy and value networks that are equivalent in architecture to applying π_θ after f_{enc}, similar to the networks in our IVG approach (to ensure a fair comparison).

For the state-based baselines we concatenated the true object positions, velocities and orientations (see table 2) to the proprioceptive robot features (see table 1, left). This is fed as input to 3-layer MLP policy networks (ELU activations, 200 hidden units each, layer normalization [14] after the first layer) and 3-layer MLP Q-value networks (ELU activations, 300 units each, layer normalization [14] after the first layer) which additionally take the actions (concatenated to other features) as input. To train SVG(0) we used the same relative entropy regularization technique as in for our method (using $\lambda = 10^{-3}$). For MPO we used the hyper-parameters from [6], which performed well across all our tests. We tuned the learning rate for both MPO and SVG(0) for performance; a rate of 1e-4 worked best.

To ensure a fair comparison between algorithms in an asynchronous setting, we ensured all algorithms ran at the same frequency of learning steps per second (which we set to 10).

F.5 Additional Baselines CEM and PG

CEM To demonstrate the value of a parametric policy we ablate it and combine a model pre-trained with our approach with an implementation of the cross-entropy method (CEM). We bootstrap with a learned value estimate as in IVG (learning this value function in the transfer learning setting as in IVG) and perform latent rollouts. In particular during training on a transfer task we replace the parametric policy with an optimization based approach using the cross-entropy method [16]. We use CEM both for computiong actions in the value function learning step and when interacting with the system. In either case the length of the rollouts for CEM is set to 5 and we use 100 trajectories in each optimization step (repeating for a total of ten steps of optimization).

PG For the likelihood ratio policy gradient baseline we use the exact same model and policy structure as for IVG. The only difference is in the calculation of the gradient of the state-action value (wrt. the policy parameters). In particular, we replace the value gradient from Equation 8 with a likelihood ratio calculation [17, 18] (using 100 rollouts for the gradient estimation).

F.6 Transfer experiment setup

We use the following three-step procedure for transferring our models to new tasks:

1. We first train the entire system from scratch (IVG) on a source task (or) a set of source tasks in the multi-task setting. From the trained modules, we choose the following model components: the encoder f_{enc}, transition model f_{trans}, and decoder f_{dec}. Only these components are transferred to the target task.

2. We initialize the parameters of the encoder, transition model and decoder using the pre-trained networks. The parameters of the policy, value and reward functions are initialized to their default values; we train these from scratch.

3. We train IVG in the usual fashion on the target task. An important point to note is that the encoder, transition and decoder networks are fine-tuned on the target task; they just have a significantly better initialization (that is generalizable to the target task). This is the primary contribution to our increase in learning speed in the transfer setting, particularly when using a model that has been trained on multiple source tasks.

G Additional Experimental results

We present a few additional experimental results in this section.

G.1 Reconstructions

Fig.8 shows a 45-step open loop rollout from an IVG(5) model, trained on the *Multitask* setting, from scratch. Even when tested on significantly longer sequences than it was trained on, the model predictions remain consistent, capturing salient details even after an open-loop prediction horizon of 30 frames. This also highlights an important point; even though the decoder predictions are not of high quality, the learned latent space can be used for robust control as it captures high-level task relevant details fairly well. Similarly, Fig.9 shows the result of a 45-step open loop prediction of proprioceptive features; the predictions are largely consistent with the observed results showing the strength of the learned model.

References

[1] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa. Learning continuous control policies by stochastic value gradients. In *NeurIPS*, 2015.

[2] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2013.

[3] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[4] D. Hafner, T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

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

[6] A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller. Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*, 2018.

[7] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[10] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[11] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Van de Wiele, V. Mnih, et al. Learning by playing-solving sparse reward tasks from scratch. In *ICML*, 2018.

[12] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[13] Z. Wojna, V. Ferrari, S. Guadarrama, N. Silberman, L.-C. Chen, A. Fathi, and J. Uijlings. The devil is in the decoder. *arXiv preprint arXiv:1707.05847*, 2017.

[14] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[15] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[16] R. Y. Rubinstein and D. P. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer, 2013.

[17] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[18] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

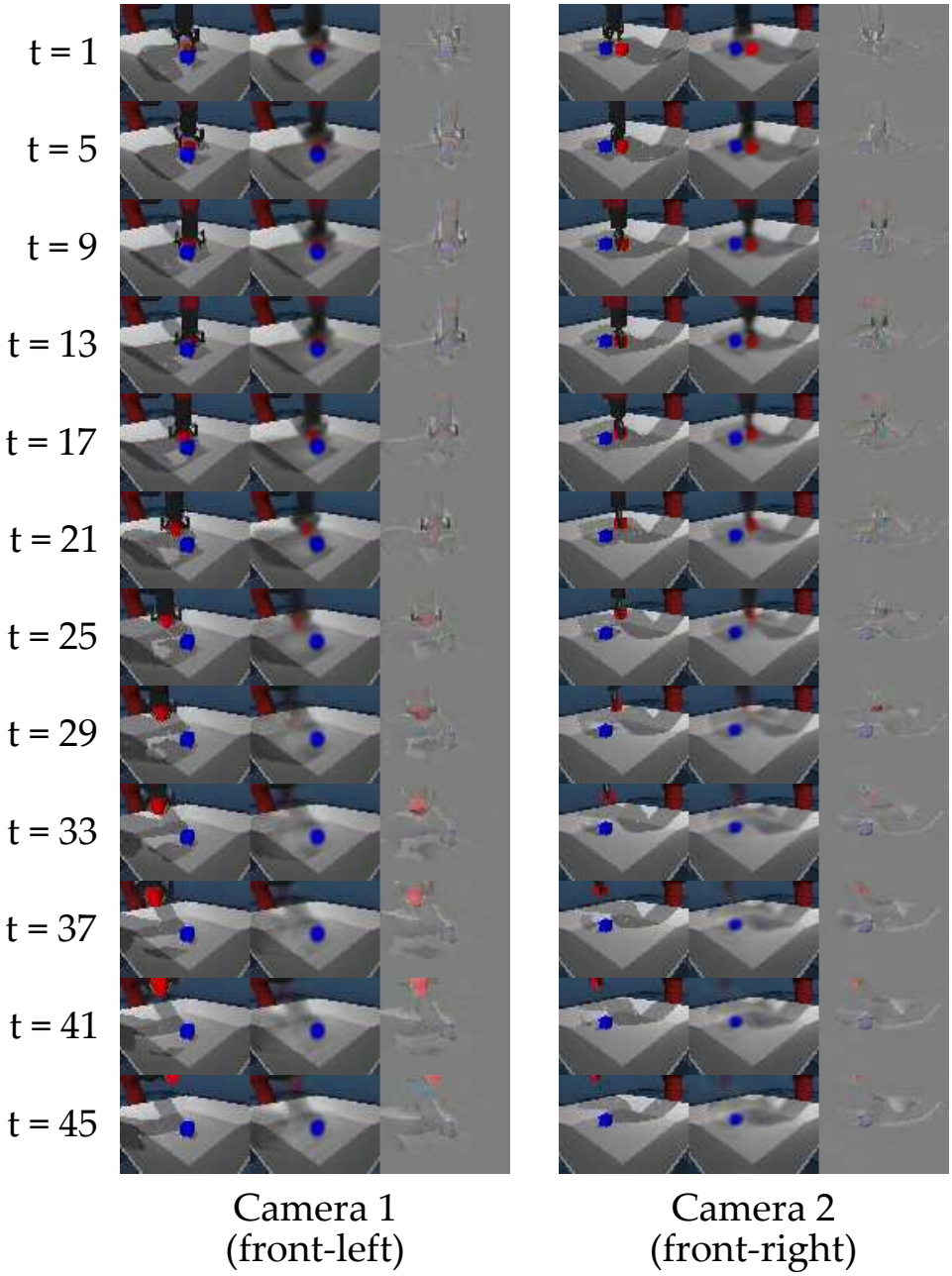


Figure 8: Example from the Lift-R task, showing open-loop reconstructions predicted by an IVG(5) model trained on the multi-task setting. The model gets a history of H observations and encodes it into the latent state $\mathbf{h}^H$ via the encoder f_{enc}. Next, it uses the sequence of actions $a^{H+1:H+N}$ to generate an open loop rollout through the transition model f_{trans}; this generates the transition states $\mathbf{h}^{H+1:H+N}$. From these states, we run the decoder f_{dec} to generate the image observations which are shown in this figure. *Left*: Images from the left camera, reconstructions and error. *Right*: Images from the right camera, reconstructions and error. For generating the predicted image sequence shown above, we set H=3 and N=45. The model was trained using IVG(5) i.e. N=5 in training. Even when running the model for significantly longer sequences than it was trained on, the predictions are consistent; the arm and objects are predicted well, albeit blurry, till around 30 frames. After 30 frames, the red object is poorly reconstructed (see the large errors near the object) but the arm positions and the blue block are still well predicted.

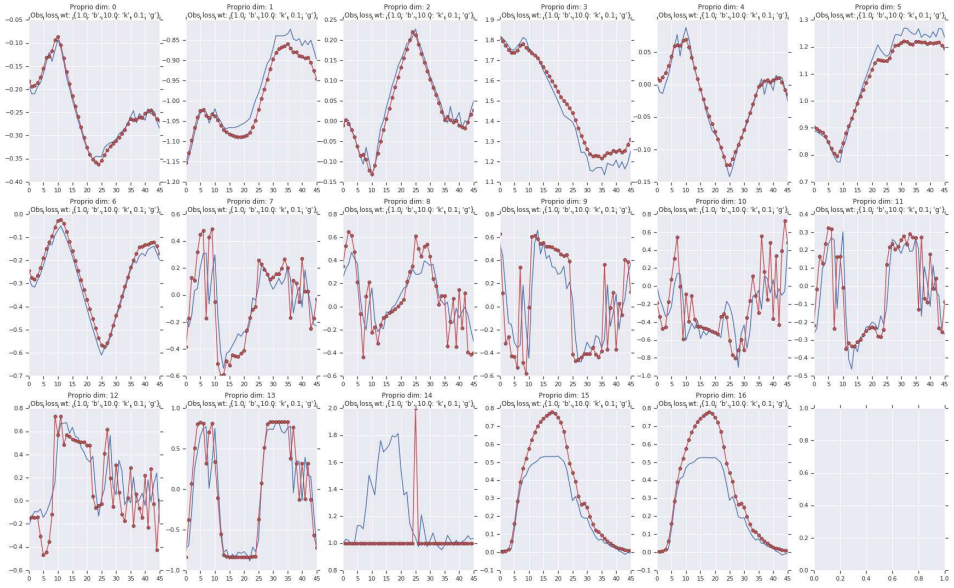


Figure 9: Example showing open-loop proprioception predictions from an IVG(5) model trained on the multi-task setting. Similar to the image reconstruction setting, we use a sequence of $H = 3$ observations and an open loop sequence of $N = 45$ actions ($\mathbf{a}^{H+1:H+N}$) to generate the proprioception predictions (blue line). The targets are plotted in red. The predictions from the IVG model are largely consistent, even for dimensions that are highly noisy; the learned latent state is able to encode the proprioceptive features and the transition model can recover their dynamics well.