Bayesian Optimization in Variational Latent Spaces with Dynamic Compression

Rika Antonova*
KTH Royal Institute of Technology, Sweden
antonova@kth.se

Akshara Rai*
Facebook AI Research
akshararai@fb.com

Tianyu Li
Facebook AI Research

Danica Kragic
KTH Royal Institute of Technology, Sweden

Abstract: Data-efficiency is crucial for autonomous robots to adapt to new tasks and environments. In this work, we focus on robotics problems with a budget of only 10-20 trials. This is a very challenging setting even for data-efficient approaches like Bayesian optimization (BO), especially when optimizing higher-dimensional controllers. Previous work extracted expert-designed low-dimensional features from simulation trajectories to construct informed kernels and run ultra sample-efficient BO on hardware. We remove the need for expert-designed features by proposing a model and architecture for a sequential variational autoencoder that embeds the space of simulated trajectories into a lower-dimensional space of latent paths in an unsupervised way. We further compress the search space for BO by reducing exploration in parts of the state space that are undesirable, without requiring explicit constraints on controller parameters. We validate our approach with hardware experiments on a Daisy hexapod robot and an ABB Yumi manipulator. We also present simulation experiments with further comparisons to several baselines on Daisy and two manipulators. Our experiments indicate the proposed trajectory-based kernel with dynamic compression can offer ultra data-efficient optimization.

Keywords: Bayesian Optimization, Data-efficient Reinforcement Learning, Variational Inference

1 Introduction

Reinforcement learning (RL) is becoming popular in robotics, since in some cases it can deal with real-world challenges, such as noise in control and measurements, non-convexity and discontinuities in objectives. However, most flexible RL methods require thousands to millions of data samples, which can make direct application to real-world robotics infeasible. For example, 10,000 30s trials/episodes on a real robot would require $\approx$100 hours of operation. Most full-scale platforms, especially in locomotion, cannot operate this long without maintenance. Nowadays, commercially available arms can operate for longer, however sophisticated anthropomorphic hands and advanced grippers are still highly prone to breakage after even a handful of trials [1]. Hence the need for algorithms that can learn in very few trials, without causing significant wear-and tear to the hardware.

In this work we focus on cases with a budget of only 10-20 trials. In such settings, using approaches like Bayesian optimization (BO) to adjust parameters of structured controllers can help improve data efficiency. However, success of BO on hardware has been demonstrated either with low-dimensional controllers or with simulation-based kernels that required hand-designed features. We propose learning simulation-based kernels in an unsupervised way with a sequential variational autoencoder (SVAE). Our approach embeds simulated trajectories $\boldsymbol{\xi}$ to a space of latent paths $\boldsymbol{\tau}$, and jointly learns a probability distribution $p(\boldsymbol{\tau}|\boldsymbol{x})$ that controllers with parameters $\boldsymbol{x}$ induce over the space of latent paths. We were inspired by initial success of trajectory-based BO kernels [2], however that was demonstrated for BO in low dimensions (2-4D). Our results show that performance of

*Both of these authors contributed equally.

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

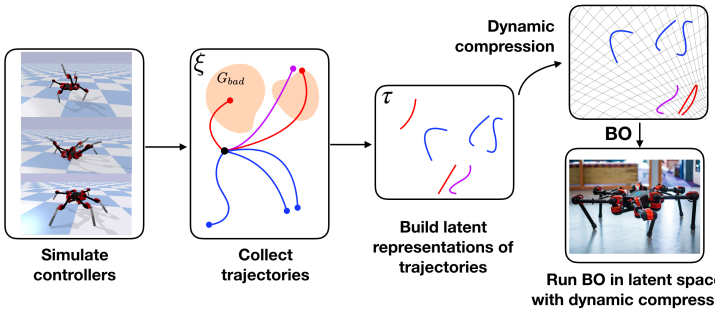

Figure 1: An overview of our approach: We start by simulating controllers and collecting their trajectories $\boldsymbol{\xi}$, along with the fraction of time spent in undesirable regions given by G_{bad}. Next, we learn to embed trajectories into a lower-dimensional a space of latent paths $\boldsymbol{\tau}$. We use dynamic compression to scale distances between latent paths based on their desirability. This dynamically compressed latent space is used for BO on hardware. Trajectory data $\boldsymbol{\xi}$ consists of high-frequency readings of robot joint angles and object position/velocity estimates (the framework can accommodate vision-based data in the future, but we do not experiment with it in this work).

a kernel based on raw trajectories deteriorates quickly for higher-dimensional problems. In contrast, our kernel based on latent paths can still offer gains even for 48-dimensional controllers.

Global optimization in latent space can still suffer from sampling unsuccessful controllers, especially in the absence of dense rewards. One solution can be adding domain-specific constraints to point optimization in the right direction. While these can be hard to define in controller parameter space, frequently they can be easily expressed in observation/state space. For example, high velocities might be undesirable if they result in hard impacts. However, formulating this as constrained optimization could result in overly conservative controllers. Instead, we incorporate controller desirability into BO by reducing exploration in the part of the trajectory space that leads to undesirable behavior. We compress the search space during BO dynamically by scaling the distance between controllers based on their desirability, initially inferred from simulation. BO can then quickly reject the undesirable parts of the search space, allowing for more exploration in the desirable parts. Figure 1 gives an overview of the proposed approach.

We test our approach (SVAE-DC: informed SVAE kernel with Dynamic Compression) on a Daisy hexapod and an ABB Yumi manipulator on hardware. We also conduct further simulation-based analysis on Daisy and two manipulators. On Daisy, our method consistently learns to walk in less than 10 hardware trials, outperforming uninformed BO. We also demonstrate significant gains on a nonprehensile manipulation task on Yumi. All latent components of our kernel can be adjusted online (by optimizing marginal likelihood as is done for BO hyperparameters). We anticipate that such adjustment could be useful for future works for settings with a medium budget of trials ($\approx$100+). Our code builds on the recently released BoTorch library [3] that supports highly scalable BO on GPUs. We open source our code for simulation environments, training and BO[1].

2 Background and Related Work

For learning with a small number of trials we turn to Bayesian Optimization (BO). It can be thought of as a data-efficient RL method that obtains a reward only at the end of each trial/episode. BO offers a principled way to trade-off exploration vs exploitation (see BO introduction and overview in [4]). For higher-dimensional robotics problems BO can benefit significantly from using simulation-based kernels. However, previous work required defining domain-specific features to be extracted from large-scale simulation data (see Section 2.1). Variational Autoencoders (VAEs) [5] provide an unsupervised alternative for embedding high-dimensional observations into a lower-dimensional space. For example, [6] recently used VAE in a Gaussian Process (GP) kernel to optimize chemical molecules. In robotics, VAEs have been used to process visual and tactile data (see [7] for a survey). We are interested in encoding trajectory data, so a sequential VAE (SVAE) could be applicable. [8, 9] show SVAEs learning latent dynamics. However, their physics simulations are low-dimensional (e.g. position of a 2D ball), sequences have length 20-30 steps, and the focus is on visual reconstruction. We aim to develop SVAE architecture that can easily handle simulations from full-scale robotics systems (state spaces 27D+) and much longer sequences (lengths 500-1000).

Our original motivation for embedding trajectory data into the kernel was Behavior Based Kernel (BBK) [2]. On low-dimensional problems it outperformed PILCO [10], which is one of the most

[1] SVAE-DC and BO code: `https://github.com/contactrika/bo-svae-dc`

popular model-based RL algorithms and has been widely used for small domains. For larger domains, such as those in our experiments, scaling PILCO can be difficult or intractable (see Section 5 in [2]). Instead of a direct comparison to PILCO, we compare our approach to a scalable version of BBK. BBK is directly applicable only to stochastic policies, but we adapted it to our setting as BBK-KL baseline. We randomize simulator parameters when collecting trajectories. Hence, even if the simulator and controllers are deterministic, each controller still induces a probability distribution over the trajectories. As proposed for BBK, for kernel distances we used symmetrized KL between trajectory distributions induced by the controllers. The generation and reconstruction parts of SVAE were used to estimate this KL. Since this baseline uses a neural network in the kernel, there is some relation to methods in [11, 12] (though these focused on GP regression, and did not use trajectories).

A part of our work can be viewed as learning a low-dimensional representation of trajectories, which is widely studied in robotics. For example, [13] use dynamic movement primitives (DMPs) to encode human demonstrations. Our locomotion controller is a variant of a cyclic DMP, which assumes synchronization between the different joints of the robot. For locomotion, we provide comparisons to BO with a standard kernel, which gives a sense of the performance of optimizing DMP parameters with standard BO. However, for manipulation DMPs require demonstrations for data-efficiency. Since we do not assume access to those, such approaches cannot be directly compared to our setup.

2.1 BO for Locomotion and Manipulation

Locomotion controllers most commonly used for real systems are structured and parametric [14, 15, 16]. BO has been used to optimize their parameters, e.g. [17, 18, 19]. Typically, these methods take $\approx$40 trials for low-dimensional controllers (3-5D). For high-dimensional controllers further domain information is needed. For example [20] use simulation and user-defined features to transform the space of a 36-dimensional controller into 6D, making the search for walking controllers of a hexapod much more data-efficient. [21] employ bipedal locomotion features to build informed kernels. While a number of other RL methods can succeed in simulation, obtaining results applicable for locomotion on hardware is challenging. Recently, [22, 23] showed that a deep RL method (PPO [24]) can be used for locomotion on hardware. However, they learn conservative controllers in simulation and help transfer via system identification of actuator dynamics [22] and a user-designed structured controller [23]. While these methods can help, they do not guarantee that a controller learned in simulation will perform well on hardware. [25] showed learning to walk on a Minitaur quadruped in only two hours. Minitaur has 8 motors that control its longitudinal motion, and no actuation for lateral movements. In comparison, our hexapod (Daisy) has 18 motors and omni-directional movements. Hence, learning control for Daisy would require significantly longer training. Since most present day locomotion robots (including Daisy) get damaged from wear and tear when operated for long, approaches that succeed for simpler quadruped controllers could be intractable in this setting.

In manipulation, active learning and BO have been used, for example, for grasping [26, 27]. These works did not incorporate simulation into the kernel, so their performance would be similar to BO with uninformed/standard kernel. [28] showed advantages of a simulation-based kernel, but needed grasping-specific features. Somewhat related are works in sim-to-real transfer, like [1], though many have visuomotor control as the focus (not considered here) and usually do not adapt online. [29] do adjust simulation parameters to match reality, so it would be interesting to combine this with BO in the future for global optimality (their work employs PPO, which is locally optimal). Due to uncertainty over friction and contact forces, sim-to-real is challenging for non-prehensile problems. However, such motions can be useful to make solutions feasible (e.g pushing when the object is too large/heavy to lift or the goal is out of reach). [30, 31] report success in transfer/adaptation on a push-to-goal task, showing the task is challenging but feasible. In our experiments we consider a 'stable push' task: push two tall objects across a table without tipping them over. The further challenges come from interaction between objects and inability to recover from them tipping over.

3 SVAE-DC: Learning Informed Trajectory-based Embeddings

We model our setting as a joint Variational Inference problem: learning to compress/reconstruct trajectories while at the same time learning to associate controllers with their corresponding probability distributions over the latent paths. For this we develop a version of sequential VAE (SVAE). The training is guided by ELBO (Evidence Lower Bound) derived for our setting directly from the modeling assumptions and doesn't require any auxiliary objectives. First, we define notation:

$\pi_{\boldsymbol{x}}$: policy/controller with parameters $\boldsymbol{x}, \boldsymbol{x} \in \mathbb{R}^D$; policies can be either deterministic or stochastic; for brevity we will refer to $\pi_{\boldsymbol{x}}$ simply as 'controller $\boldsymbol{x}$'

$\boldsymbol{\xi} \equiv \boldsymbol{\xi}_{1:T}$: original trajectory for T time steps containing high-frequency sensor readings

$\boldsymbol{\tau} \equiv \boldsymbol{\tau}_{1:K}$: latent space 'path' (embedding of a trajectory)

$p(\boldsymbol{\xi}_{1:T}|\boldsymbol{x})$: a conditional probability distribution over the trajectories induced by controller $\boldsymbol{x}$; the relationship between the controller and trajectories could be probabilistic either because the controller is stochastic, or because the simulator environment is stochastic, or both

$p(\boldsymbol{\tau}_{1:K}|\boldsymbol{x})$: a conditional probability distribution over latent space paths induced controller by $\boldsymbol{x}$

$G_{bad} : S \to \{0, 1\}$ a map denoting whether an observation $\boldsymbol{\xi}_t \in S$ is within an undesirable region

y : fraction of time $\boldsymbol{\xi}$ spends in undesirable regions; $\boldsymbol{\psi}$ learns analogous notion for a latent path $\boldsymbol{\tau}$

Our goal is to learn $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$. $p(\boldsymbol{\tau}|\boldsymbol{x})$ is analogous to $p(\boldsymbol{\xi}|\boldsymbol{x})$, only the paths are encoded in a lower-dimensional latent space. This is useful for constructing kernels for efficient BO on hardware. As a measure of trajectory 'quality' we can keep track of how long each trajectory spends in undesirable regions (y). For the latent paths we learn the analogous notion ($\boldsymbol{\psi} = \psi_{1:K}$), which enables modularity and fast online updates (discussed in Section 4). We do not impose hard constraints during optimization, so G_{bad} used to compute y can be specified roughly, with approximate guesses. Our framework also supports $G_{bad} : S \to [0, 1]$, but for users it is frequently easier to make a rough thresholded estimate rather than providing smooth estimates or probabilities. The graphical model we construct for this setting is shown in Figure 2. Not all independencies are captured by the illustration. So, explicitly, the generative model is: $p_{\boldsymbol{w}}(\boldsymbol{\tau}, \boldsymbol{\psi}, \boldsymbol{\xi}, y \mid \boldsymbol{x}) = p(\boldsymbol{\tau}_{1:K}, \boldsymbol{\psi}|\boldsymbol{x})p(y|\boldsymbol{\psi}) \prod_{t=1}^{T} p(\boldsymbol{\xi}_t|\boldsymbol{\xi}_{t-1}, \boldsymbol{\tau}_{1:K})$. Approximate posterior is modeled by: $q_{\boldsymbol{\phi}}(\boldsymbol{\tau}, \boldsymbol{\psi}, \boldsymbol{\xi}, y) = q(\boldsymbol{\tau}_{1:K}, \boldsymbol{\psi}|\boldsymbol{\xi}_{1:T}, y)$.

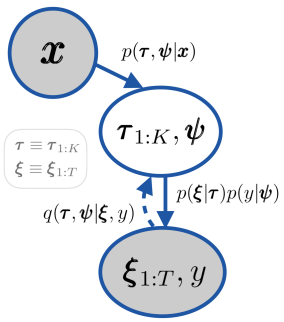

Figure 2: A sketch of generative and inference model.

We collect trajectories $\boldsymbol{\xi}_{1:T}^{(i)}$ by simulating N controllers with parameters $\boldsymbol{x}^{(i)}$ for T time steps. We derive ELBO for this setting to maximize $\log p(Data) = \log p(\{\boldsymbol{x}^{(i)}, \boldsymbol{\xi}_{1:T}^{(i)}\}_{i=1...N})$. Using '~' over the variables to indicate samples from the current variational approximation, we get:

$$\mathcal{L}^{DC}(\boldsymbol{w}, \boldsymbol{\phi}|\boldsymbol{x}, \boldsymbol{\xi}, y) = \mathbb{E}_{\substack{\tilde{\boldsymbol{\tau}}, \tilde{\boldsymbol{\psi}} \sim \\ q(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{\xi}, y)}} \Big[\log p(\boldsymbol{\xi}|\tilde{\boldsymbol{\tau}}) + \log p(y|\tilde{\boldsymbol{\psi}}) + \log p(\tilde{\boldsymbol{\tau}}, \tilde{\boldsymbol{\psi}}|\boldsymbol{x}) - \log q(\tilde{\boldsymbol{\tau}}, \tilde{\boldsymbol{\psi}}|\boldsymbol{\xi}, y) \Big] \quad (1)$$

$\boldsymbol{w}$, $\boldsymbol{\phi}$ are weights of deep neural networks optimized by gradient ascent on the ELBO.

4 Bayesian Optimization with Dynamic Compression

In Bayesian Optimization (BO), the problem of optimizing controllers is viewed as finding controller parameters $\boldsymbol{x}^*$ that optimize some objective function $f(\boldsymbol{x})$: $f(\boldsymbol{x}^*) = \max_{\boldsymbol{x}} f(\boldsymbol{x})$. At each optimization trial BO optimizes an auxiliary function to select the next promising $\boldsymbol{x}$ to evaluate. f is commonly modeled with a Gaussian process (GP): $f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j))$.

The key object is the kernel function $k(\cdot, \cdot)$, which encodes similarity between inputs. If $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is large for inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$, then $f(\boldsymbol{x}_i)$ strongly influences $f(\boldsymbol{x}_j)$. One of the most widely used kernel functions is the Squared Exponential (SE) kernel: $k_{SE}(\boldsymbol{r} \equiv |\boldsymbol{x}_i - \boldsymbol{x}_j|) = \sigma_k^2 \exp\left(-\frac{1}{2}\boldsymbol{r}^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}\right)$, where σ_k^2, $\boldsymbol{\ell}$ are signal variance and a vector of length scales respectively. σ_k^2, $\boldsymbol{\ell}$ are called 'hyperparameters' and are optimized automatically by maximizing marginal likelihood ([4], Section V-A). SE belongs to a broader class of Matérn kernels. One common parameter choice yields Matérn$_{5/2}$: $k_{\text{Matérn}_{5/2}}(\boldsymbol{r}) = \left(1 + \frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}} + \frac{5\boldsymbol{r}^2}{3\boldsymbol{\ell}^2}\right) \exp\left(-\frac{\sqrt{5}\boldsymbol{r}}{\boldsymbol{\ell}}\right)$. SE and Matérn kernels are stationary, since they depend on $\boldsymbol{r} \equiv \boldsymbol{x}_i - \boldsymbol{x}_j \; \forall \boldsymbol{x}_{i,j}$, and not on individual $\boldsymbol{x}_i, \boldsymbol{x}_j$. Section 2.1 discussed recent work that showed how to effectively remove stationarity by using informed feature transforms for kernel computations. But these required extracting domain-specific features manually, or learning to fit a pre-defined set of features using a deterministic NN in a supervised way.

We propose to use $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$ learned by SVAE-DC. [2] showed that a 'symmetrization' of KL divergence can be used to define a KL-based kernel for trajectories in the original space:

$$k_{KL} = \exp(\text{-}\alpha D(\boldsymbol{x}_i, \boldsymbol{x}_j)) \; ; \; D(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{KL(p(\boldsymbol{\xi}|\boldsymbol{x}_i)||p(\boldsymbol{\xi}|\boldsymbol{x}_j))} + \sqrt{KL(p(\boldsymbol{\xi}|\boldsymbol{x}_j)||p(\boldsymbol{\xi}|\boldsymbol{x}_i))} \quad (2)$$

In theory, we could use this to define an analogous kernel in the latent space:

$$k_{LKL} = \exp(\text{-}\alpha D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j)); D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{KL(p(\boldsymbol{\tau}|\boldsymbol{x}_i)||p(\boldsymbol{\tau}|\boldsymbol{x}_j))} + \sqrt{KL(p(\boldsymbol{\tau}|\boldsymbol{x}_j)||p(\boldsymbol{\tau}|\boldsymbol{x}_i))}$$

However, variational inference (VI) tends to under-estimate variances [32, 33, 34, 35]. Hence, our kernel works with latent means $\bar{\boldsymbol{\tau}}_{\boldsymbol{x}}, \bar{\boldsymbol{\psi}}_{\boldsymbol{x}} = E\big[p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})\big]$ directly. We define our kernel function with:

$$\boldsymbol{r}_\tau = D_\tau(\boldsymbol{x}_i, \boldsymbol{x}_j) = \left|(1-\bar{y}_{\boldsymbol{x}_i})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - (1-\bar{y}_{\boldsymbol{x}_j})\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j}\right| \;\; ; \quad y_{\boldsymbol{x}} \sim p(y|\bar{\boldsymbol{\psi}}_{\boldsymbol{x}}) \tag{3}$$

$$k_{SVAE\text{-}DC}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\left(-\tfrac{1}{2}\boldsymbol{r}_\tau^T \operatorname{diag}(\boldsymbol{\ell})^{-2}\boldsymbol{r}_\tau\right) \tag{4}$$

The form of Equation 4 allows us to apply existing machinery for optimizing kernel hyperparameters $\sigma_k^2, \boldsymbol{\ell}$ directly to the SVAE-DC kernel. Note that $\operatorname{diag}(\boldsymbol{\ell})^{-2}$ is related to covariance in the case diagonal Gaussians; L1 and L2 norms are related by $L2 \leq L1 \leq \sqrt{\dim} L2$. So BO with $|\bar{\boldsymbol{\tau}}_{\boldsymbol{x}_i} - \bar{\boldsymbol{\tau}}_{\boldsymbol{x}_j}|$ in the kernel is related to using KL in the case of diagonal Gaussians (with a simplification to capture variance-only terms by learning σ_k^2). We can also conveniently obtain SVAE-DC-Matérn version of the kernel by simply changing the form of Equation 4 to the Matérn function.

Scaling latent representations by $1-\bar{y}_{\boldsymbol{x}}$ yields dynamic compression: latent representations that correspond to controllers frequently visiting undesirable parts of the space are scaled down. With this, we retain trajectory-based distance in the desirable parts of the space, but compress it in undesirable parts to reduce unwanted exploration. The 'dynamic compression' transformation is applied after SVAE training, in addition to the compression obtained by SVAE. The scaling can be made non-linear with $sigmoid(\alpha(\bar{y}_{\boldsymbol{x}} - c))$. This achieves aggressive compression in settings with an extremely small budget of trials. The additional parameters α, c, as well as $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x}), p(y|\boldsymbol{\psi})$ can be optimized online (as BO hyperparameters). Note that because of the multiplicative formulation, two controllers with different $\boldsymbol{\tau}$ and y can appear similar during optimization. Theoretically, this can also bring the undesirable space close to a part of desirable space. We address this by updating the learned components online via GP's marginal likelihood, as in Deep kernel learning [11, 12]. However, for large NNs such online updates would only be useful after a large number of hardware trials. Hence, we provide a modular architecture to ensure that the multiplicative factors can be updated faster. We structure SVAE to learn $p(y|\boldsymbol{\psi})$ and $p(\boldsymbol{\tau}, \boldsymbol{\psi}|\boldsymbol{x})$, instead of a joint $p(y, \boldsymbol{\tau}|\boldsymbol{x})$. This makes the NN for $p(y|\boldsymbol{\psi})$ small, facilitating more data-efficient NN updates during BO. Now, during hardware trials, shifts in $\bar{y}$ will be more pronounced, compared to updates in the full latent path representation.

In summary, SVAE-DC and the resulting kernel result in a fully automatic way of learning latent trajectory embeddings in unsupervised way. For domains where G_{bad} is given, we can also achieve dynamic compression of the latent space, making BO ultra data-efficient. All the components used during BO can be optimized online via the same methods as those for adjusting BO hyperparameters.

5 SVAE-DC: NN Architecture and Training

We propose to use time convolution architecture for $q(\boldsymbol{\tau}|\boldsymbol{\xi})$, de-convolutions for $p(\boldsymbol{\xi}|\boldsymbol{\tau})$. For this we use 1D convolutions for the sequential dimensions t, k and treat the dimensions of $\boldsymbol{\xi}_t, \boldsymbol{\tau}_k$ as different channels. With that, for all our experiments (all different robot and controller architectures) we were able to use the same network parameters: 3-layer 1D convolutions with [32, 64, 128] channels (reverse order for de-convolutions; kernel size 4, stride 2) followed by MLP layer for μ, σ outputs. We were also able to use same latent space sizes: 3-dimensional $\boldsymbol{\tau}$, latent sequence length K=3 for all our experiments. This yielded a small 9D optimization space for BO, which is highly desirable for optimization with few trials. Notably, this NN architecture also retained good reconstruction accuracy, not far from results with larger latent spaces ($\boldsymbol{\tau}$=$6D, 12D$; K=$5, 15$) and hidden sizes (256-1024). We also used de-convolutional architecture for $p(\boldsymbol{\tau}|\boldsymbol{x})$. Since $p(\boldsymbol{\tau}|\boldsymbol{x})$ was one of the key parts for BO we used 4 layers with [512, 256, 128, 128] channels (though a smaller CNN could have sufficed). For $p(y|\psi)$ we used a 2-layer MLP (hidden size 64). Training took ≈30-180 minutes on 1 GPU, using $1e$-4 learning rate (decayed to $1e$-5). We note that other advanced architectures like RNNs, LSTMs and Quasi-RNNs [36] did not result in reliably robust training in our experiments.

6 Locomotion Experiments on the Daisy Hexapod

For locomotion experiments, we use a Daisy robot (Figure 3) from Hebi robotics [37]. It has six legs, each with 3 motors – base, shoulder and elbow. A Vive tracking system measures the robot's

position in a global frame for rewards. To obtain simulated trajectories for training SVAE we used PyBullet [38]. The simulator was fast, but did not have an accurate contact model with the ground. While free-space motion of individual joints transferred to hardware, the overall behavior of the robot when interacting with the ground was very different between simulation and hardware. As a result, rewards obtained by controllers in simulation could be significantly different on hardware.

Daisy Controllers: We used Central Pattern Generators (CPGs) from [39]. CPGs are a variant of rhythmic DMPS [13], capable of generating a large number of locomotion gaits by changing the frequency, amplitude, and offset of each joint, as well as the relative phase differences between joints. Different CPG parameters can be restricted to obtain controllers with various dimensionalities. We experimented with 11D controller on hardware and 27D in simulation. For hardware, we assume that all joints have the same amplitude, frequency and offset (3 parameters), all base motors have independent phases (6 parameters), all shoulders and elbows have the same phase difference w.r.t. the base (2 parameters). This assumption implies that all joints are treated identically, which doesn't always hold, since each motor has slightly different tracking and bandwidth. In the future, we would like to use alternatives that allow each motor to learn independently. For simulation: base, shoulder and elbow joints were allowed to have independent amplitudes, frequencies and offsets, but fixed across the six legs (9 parameters); each of the 18 joints was allowed to have an independent phase (18 parameters).

Figure 3: Daisy hexapod used in this work.

Daisy Hardware Experiments: To construct SVAE-DC kernel for BO we trained SVAE using $500,000$ simulated trajectories (1000 time steps each, $\approx 16.5s$). For dynamic compression the states were marked as undesirable if they had: high joint velocities (more than 10rad/sec); robot base tilting by more than 60°in roll and pitch, elbows hitting the ground; height of the base outside of [0.1, 0.7]cm from the ground. These aimed to reduce the chance of the robot breaking: controllers with high joint velocities can harm the motors on impact with the ground; tilting the torso can cause the robot to fall on its back; scraping the ground or lifting off and then falling can cause further damage. Since our BO trials were in a narrow walkway, we marked as undesirable states deviating more than 0.5m from the starting x-coordinate of the base.
The objective for BO was: $f(\boldsymbol{x}) = 10 \cdot y_{final} - N_{high\_vel}$, where y_{final} was the final y-coordinate of the robot (how much the robot walked forward), N_{high_vel} was the number of timesteps with velocities exceeding 10rad/sec. All BO experiments used UCB acquisition function (with $\beta = 1$). We completed 5 runs of BO on the Daisy robot hardware, initializing with 2 random samples, followed by 10 trials of BO (Figure 4). We also conducted baseline experiments with SE kernel by directly searching in the space of CPG parameters. This served as a comparison to more traditional trajectory compression methods that optimize DMPs (since CPG can be seen as a DMP variant). For Daisy robot, the controller would be considered acceptable if it walked forward for more than $1.5m$ during a trial of 25 seconds on hardware. For comparison to random search we sampled 60 controllers at random. Of these only 2 were able to walk forward a distance of over $1.5m$ in $25s$. So the problem was challenging, as the chance of randomly sampling a successful controller was $<4\%$. BO with SVAE-DC kernel found walking controllers reliably in all 5/5 runs within fewer than 10 trials. In contrast, both BO with SE found forward walking controllers only in 2/5 runs.

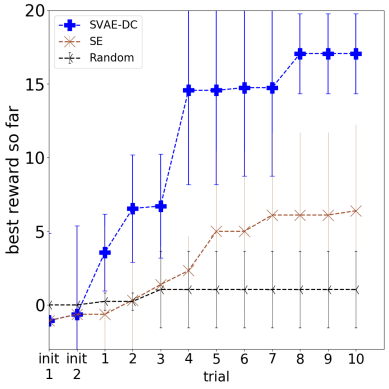

Figure 4: BO on Daisy hardware (means over 5 runs, 90% CIs).

Further experiments in simulation: We created an artificial 'sim-to-real' gap, allowing to gauge the potential for simulation-based kernels without running all the experiments on hardware. For each BO run we randomly sampled ground restitution parameters, and kept them fixed for all trials within a run. Hence, simulation-based kernels did not have full information about the exact properties of the environment used during BO. The range of parameters was the same for BO and for data collection,

so informed kernels could identify controllers that perform well on average across settings. But such informed kernel could have caused negative transfer by lagging to identify controllers best for a particular BO setting, and instead favoring conservative (crawling) best-across-settings controllers. Figure 5 shows BO with 27D controller. BO with SVAE-DC outperformed all baselines. BBK-KL kernel obtained smaller improvements over SE and Random baselines. This indicated that a trajectory-based kernel was useful even when optimizing a high-dimensional controller, although BBK-KL benefits were greatly diminished compared to BBK results for 2-4 dimensional controllers reported in prior work. In these experiments, SVAE without dynamic compression was very similar to SE (omitted from the plot for clarity, since it was overlapping with SE). This showed that dimensionality reduction alone does not guarantee improvement (even when the latent space contains information needed to decode back into the space of original trajectories).

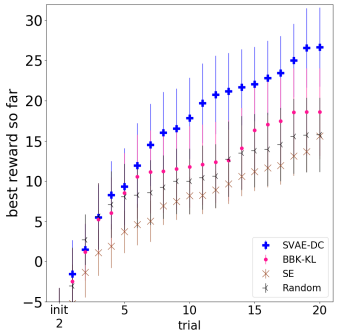

Figure 5: BO for Daisy in simulation (means over 50 runs, 90% CIs).

7 Manipulation Experiments

Our manipulation task was to push two objects from one side of the table to another without tipping them over. We used ABB Yumi robot for our hardware experiments (Figure 6), and conducted additional simulation experiments with Yumi and Franka Emika robot models. We used PyBullet for simulation. For Yumi environment the objects had mass and inertial properties similar to paper towel rolls (mass of 150g, 22cm height, 5cm radius); for Franka these had properties similar to wooden rolls (2kg, 22cm height, 8cm radius). Compared to 'push-to-target' task, our task had two different challenges. The objects were likely to come into contact with each other (not only the robot arm). Moreover, they could easily tip over, especially if forces were applied above an object's center of mass. Reward was given only at the end of the task: the distance each upright object moved in the desired direction minus a penalty for objects that tipped over (with y_{max} being table width): $f(\boldsymbol{x}) = \sum_i \left[(y^{obj_i}_{final} - y^{obj_i}_{start}) \mathbb{1}_{obj_i \in Up} - y_{max} \mathbb{1}_{obj_i \in Tipped} \right]$.

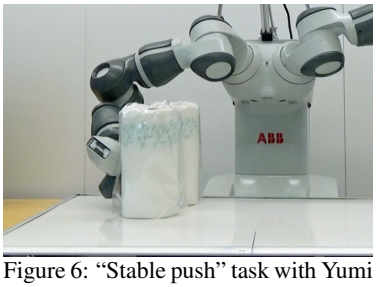

Figure 6: "Stable push" task with Yumi

Controllers: We tested our approach on two types of controllers: 1) joint velocity controller suitable for robots like ABB Yumi and 2) torque controller suitable for robots like Franka Emika. The first was parameterized by 6 joint velocity "waypoints", one target velocity for each joint of the robot arm (so $6 \cdot 7 = 42$ parameters for a 7DoF arm). Each "waypoint" also had a duration parameter that specified the fraction of time to be spent attaining the desired joint velocities. Overall this yielded a 48-dimensional parametric controller. The second controller type was aimed to be safe to use on robots with torque control that are more powerful than ABB Yumi. Instead of exploring randomly in torque space, we designed a parametric controller with desired waypoints in end-effector space. Each of the 6 waypoints had 6 parameters for the pose (3D position, 3D orientation) and 2 parameters for controller proportional and derivative gains. Overall this yielded a 48-dimensional parametric controller: $6 \cdot (6+2)$. This controller interpolated between the waypoints using a 5^{th} order minimum jerk trajectory for positions, and used linear interpolation for orientations. End effector Jacobian for the corresponding robot model was used to convert to joint torques.

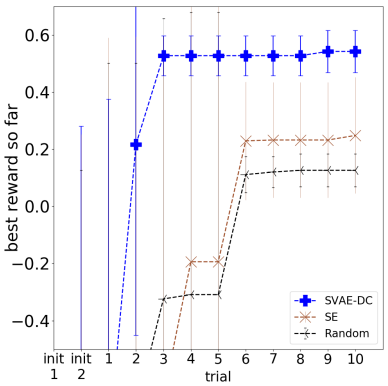

Figure 7: BO on ABB Yumi hardware (means over 5 runs, 90% CIs).

Yumi Hardware Experiments: For constructing SVAE-DC kernel used during BO on hardware we simulated 500,000 trajectories. These contained joint angles of the robot and object poses at each time step (1000 steps per trajectory). A step t on a trajectory $\boldsymbol{\xi}$ was marked as undesirable $\left(G_{bad}(\boldsymbol{\xi}_t) = 1\right)$ when: any object tipped over or was pushed beyond the table; robot collided with the table; the end effector was outside of main workspace (not over the table area).

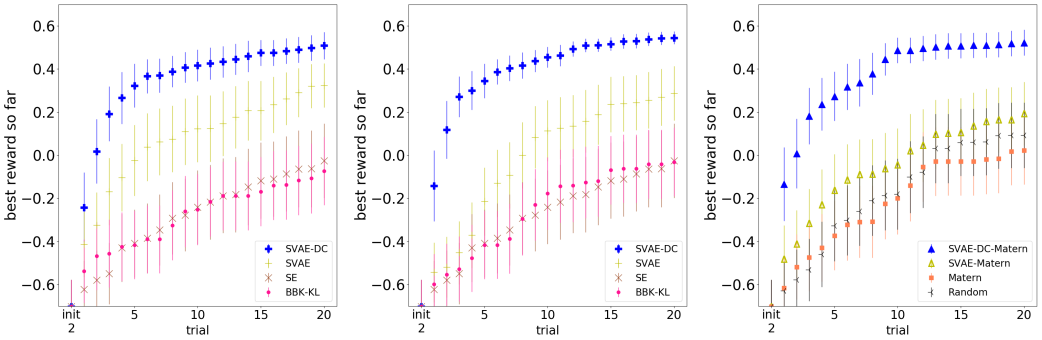

Figure 9: BO with various kernels on Franka Emika simulation. Left: SVAE trained with same parameters as in all the previous experiments. Middle: SVAE with larger latent space and NNs. Right: Matern used as outer function for all kernel. The plots show means over 50 runs, 90% CIs.

Mass, friction and restitution of the objects were randomized at the start of each episode/trajectory. Randomization ranges were set to roughly resemble variability of how real-world objects behaved. ABB Yumi robot available to us could operate effectively only at low velocities ($\frac{1}{5}$ of simulation maximum). High-velocity trajectories successful in simulation yielded different results on hardware. To prevent Yumi from shutting down due to high load we stopped execution if the robot's arm extended too far outside the main workspace, also stopped if it was about to collide with the table (giving $-2y_{max}$ reward in such cases). These factors caused a large sim-real gap. Nonetheless, BO with SVAE-DC kernel was still able to significantly outperform BO with SE (Figure 7). Even when controllers successful in simulation yielded very different outcomes on hardware, SVAE-DC kernel was still able to find well-performing alternatives (more conservative, yet successful on hardware).

Further Yumi and Franka experiments in simulation: We emulated 'sim-to-real' gap as with Daisy simulation: sampled different object properties (mass, friction, restitution) at the start of each BO run. Results in Figure 8 show that BO on Yumi with SVAE-DC kernel yielded substantial improvement over all baselines. BO in the latent space of SVAE (without dynamic compression) was also able to substantially outperform all baselines, matching SVAE-DC gains after ≈15 trials.

Figure 9 shows BO results on Franka Emika simulation (left). Kernels were built in the same way as for Yumi, but from shorter trajectories (500 steps). Furthermore, we analyze how increasing the size of SVAE latent space and NNs impacts performance (middle). The larger latent space is $6{\cdot}5{=}30$D (vs 9D in other experiments), the hidden layer size of NNs is increased from 128 to 256. Larger latent space implies larger search space for BO, which could impair data efficiency. BO with SVAE kernel (no DC) still outperforms BBK-KL and SE kernels, but only after 10 trials. BO with SVAE-DC offers immediate gains with low variance between runs (well-performing points are found more reliably). This indicates that dynamic compression could counter-balance increase in kernel dimensionality. Finally, we experimented with Matérn kernel (right plot in Figure 9), but it did not show benefits over using SE kernel. We attempted changing hyperparameter prior and restricting hyperparameter ranges, but it did not consistently outperform random search (same held for SE in high dimensions). The performance of BO with SVAE kernel using Matérn as outer kernel function showed modest improvement over baselines. In contrast, BO with SVAE-DC kernel still offered substantial improvements.

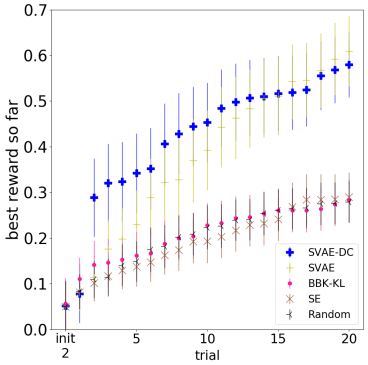

Figure 8: BO on ABB Yumi simulation (mean of 50 runs, 90% CIs).

8 Conclusion

In this work, we employed BO to optimize robot controllers with a small budget of trials. Previously, the success of BO has been either limited to low-dimensional controllers or required kernels with domain-specific features. We proposed an unsupervised alternative with sequential variational autoencoder. We used it to embed simulated trajectories into a latent space, and to jointly learn relating controllers with latent space paths they induce. Furthermore, we provided a mechanism for dynamic compression, helping BO reject undesirable regions quickly, and explore more in other regions. Our approach yielded ultra-data efficient BO in hardware experiments with hexapod locomotion and a manipulation task, using the same SVAE-DC architecture and training settings across experiments.

Acknowledgments

This research was supported in part by the Knut and Alice Wallenberg Foundation.

References

[1] OpenAI. Learning Dexterous In-Hand Manipulation. *arXiv:1808.00177*, 2018.

[2] A. Wilson, A. Fern, and P. Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *Journal of Machine Learning Research*, 15(1):253–282, 2014.

[3] M. Balandat, B. Karrer, D. Jiang, B. Letham, S. Daulton, A. Wilson, E. Bakshy. BoTorch. https://botorch.org/. Accessed: 2019-05.

[4] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[5] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv:1312.6114*, 2013.

[6] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

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

[8] L. Yingzhen and S. Mandt. Disentangled sequential autoencoder. In *International Conference on Machine Learning*, pages 5656–5665, 2018.

[9] M. Fraccaro, S. Kamronn, U. Paquet, and O. Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3601–3610, 2017.

[10] M. Deisenroth and C. E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[11] R. Calandra, J. Peters, C. E. Rasmussen, and M. P. Deisenroth. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 3338–3345. IEEE, 2016.

[12] A. G. Wilson, Z. Hu, R. Salakhutdinov, and E. P. Xing. Deep kernel learning. In *Artificial Intelligence and Statistics*, pages 370–378, 2016.

[13] P. Pastor, L. Righetti, M. Kalakrishnan, and S. Schaal. Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE, 2011.

[14] N. Thatte, H. Duan, and H. Geyer. A method for online optimization of lower limb assistive devices with high dimensional parameter spaces. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–6. IEEE, 2018.

[15] S. Feng, E. Whitman, X. Xinjilefu, and C. G. Atkeson. Optimization-based Full Body Control for the DARPA Robotics Challenge. *Journal of Field Robotics*, 32(2):293–312, 2015.

[16] Y. Gong, R. Hartley, X. Da, A. Hereid, O. Harib, J.-K. Huang, and J. Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. *arXiv:1809.07279*, 2018.

[17] R. Calandra. *Bayesian Modeling for Optimization and Control in Robotics*. PhD thesis, Darmstadt University of Technology, Germany, 2017.

[18] D. J. Lizotte, T. Wang, M. H. Bowling, and D. Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 944–949, 2007.

[19] M. Tesch, J. Schneider, and H. Choset. Using response surfaces and expected improvement to optimize snake robot gait parameters. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1069–1074. IEEE, 2011.

[20] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

[21] A. Rai, R. Antonova, F. Meier, and C. G. Atkeson. Using simulation to improve sample-efficiency of bayesian optimization for bipedal robots. *Journal of Machine Learning Research*, 20(49):1–24, 2019.

[22] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv:1804.10332*, 2018.

[23] T. Li, A. Rai, H. Geyer, and C. G. Atkeson. Using deep reinforcement learning to learn high-level policies on the atrias biped. *arXiv:1809.10811*, 2018.

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

[25] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems (RRS)*, 2019.

[26] O. Kroemer, R. Detry, J. Piater, and J. Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010.

[27] L. Montesano and M. Lopes. Active learning of visual descriptors for grasping using non-parametric smoothed beta distributions. *Robotics and Autonomous Systems*, 60(3):452–462, 2012.

[28] R. Antonova, M. Kokic, J. A. Stork, and D. Kragic. Global search with bernoulli alternation kernel for task-oriented grasping informed by simulation. In *Conference on Robot Learning*, pages 641–650, 2018.

[29] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *arXiv:1810.05687*, 2018.

[30] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[31] I. Arnekvist, D. Kragic, and J. A. Stork. VPE: Variational Policy Embedding for Transfer Reinforcement Learning. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2019.

[32] T. Minka. Divergence measures and message passing. *TR-2005-173 Microsoft Research*, 2005.

[33] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[34] C. Riquelme, M. Johnson, and M. Hoffman. Failure modes of variational inference for decision making. *Prediction and Generative Modeling in RL Workshop (AAMAS, ICML, IJCAI)*, 2018.

[35] S. Tschiatschek, K. Arulkumaran, J. Stühmer, and K. Hofmann. Variational inference for data-efficient model learning in pomdps. *TR-2018-15 Microsoft Research*, 2018.

[36] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *International Conference on Learning Representations*, 2017.

[37] Hebi Robotics. `http://docs.hebi.us`. Accessed: 2019-06.

[38] Pybullet simulator. `https://github.com/bulletphysics/bullet3`. Accessed: 2019-06.

[39] A. Crespi and A. J. Ijspeert. Online optimization of swimming and crawling in an amphibious snake robot. *IEEE Transactions on Robotics*, 24(1):75–87, 2008.