Deep Variational Koopman Models: Inferring Koopman Observations for Uncertainty-Aware Dynamics Modeling and Control

Jeremy Morton[1], **Freddie D. Witherden**[2] and **Mykel J. Kochenderfer**[1]
[1]Department of Aeronautics and Astronautics, Stanford University
[2]Department of Ocean Engineering, Texas A&M University
jmorton2@stanford.edu, fdw@tamu.edu, mykel@stanford.edu

Abstract

Koopman theory asserts that a nonlinear dynamical system can be mapped to a linear system, where the Koopman operator advances observations of the state forward in time. However, the observable functions that map states to observations are generally unknown. We introduce the Deep Variational Koopman (DVK) model, a method for inferring distributions over observations that can be propagated linearly in time. By sampling from the inferred distributions, we obtain a distribution over dynamical models, which in turn provides a distribution over possible outcomes as a modeled system advances in time. Experiments show that the DVK model is effective at long-term prediction for a variety of dynamical systems. Furthermore, we describe how to incorporate the learned models into a control framework, and demonstrate that accounting for the uncertainty present in the distribution over dynamical models enables more effective control.

1 Introduction

In order to analyze, control, and predict the evolution of dynamical systems, we require knowledge about their governing equations. For many complex systems of interest, the exact governing equations are either unknown or are prohibitively expensive to accurately evaluate. These challenges have inspired recent interest in learning system dynamics directly from data. In particular, neural network dynamics models have garnered widespread attention due to their ability to model complex functions with high-dimensional inputs such as image data [Krishnan *et al.*, 2017; Rangapuram *et al.*, 2018; Moerland *et al.*, 2017; Karl *et al.*, 2017; Fraccaro *et al.*, 2017].

Data-driven dynamics modeling is of particular interest to the field of reinforcement learning (RL), where the goal is to automatically learn control policies that satisfy predefined objectives. Model-based RL algorithms, which attempt to explicitly model environment dynamics, have the potential to solve complex tasks while requiring significantly less experience than model-free algorithms. However, the difficulty of constructing accurate data-driven dynamics models has so far allowed model-free approaches to outperform model-based approaches on many problems. Nonetheless, recent work has demonstrated that planning algorithms combined with neural network dynamics models can achieve strong performance on a variety of tasks while requiring less environmental interaction than state-of-the-art model-free RL algorithms [Chua *et al.*, 2018; Hafner *et al.*, 2018].

The exact form of a dynamics model has strong implications for how easily the model can be incorporated into a control or planning framework. Neural networks are nonlinear functions, meaning neural dynamics models might not be well suited for many control methods designed for linear systems. For this reason, approaches such as E2C [Watter *et al.*, 2015] and RCE [Banijamali *et al.*, 2018] train neural networks to map states to a latent space where the dynamics can be evolved according to locally linear models that enable action selection through iLQR. Koopman theory [Koopman, 1931] offers an alternative viewpoint through which nonlinear dynamics can be mapped to linear dynamics. It posits the existence of a linear operator that acts on observable functions of the state to advance them forward in time. The exact form of the observable functions is usually not known, but recent work has sought to learn them automatically using neural networks [Lusch *et al.*, 2018; Takeishi *et al.*, 2017; Li *et al.*, 2017; Otto and Rowley, 2017]. Furthermore, it has been shown that data-driven models that leverage Koopman theory can be used for control in a wide array of applications [Kaiser *et al.*, 2017; Korda and Mezić, 2018; Morton *et al.*, 2018].

In this work, we introduce the Deep Variational Koopman (DVK) model, a method for inferring distributions over Koopman observations that can be propagated linearly in time. Our method requires the training of a single neural network model, but enables the sampling of an ensemble of linear dynamics models in the space of observations. Taken together, this model ensemble effectively provides a distribution over the system dynamics. In evaluations on benchmark problems, we demonstrate that DVK models can be used for accurate long-term prediction with reasonable uncertainty estimates. Additionally, we explain how the linear model ensembles can be easily incorporated into an existing control framework, and we empirically demonstrate that controller effectiveness improves as the size of the model ensemble grows.

2 Dynamics Modeling

In this section, we provide background on the Koopman operator and its relation to forced dynamical systems. Subsequently,

we derive an objective function for inferring distributions over Koopman observations, which yields a practical training procedure for inferring Koopman observations from data.

2.1 The Koopman Operator

Consider a nonlinear discrete-time dynamical system subject to control inputs described by $\mathbf{x}_{t+1} = F(\mathbf{x}_t, \mathbf{u}_t)$, where $\mathbf{x}_t \in \mathbb{R}^n$ and $\mathbf{u}_t \in \mathbb{R}^p$. Koopman theory asserts that there exists an infinite-dimensional linear operator $\mathcal{K}$ that advances all observable functions h of the state and control inputs forward in time. Under the assumption that the control inputs are not evolving dynamically, this update equation takes the form [Proctor *et al.*, 2018]:

$$\mathcal{K}h(\mathbf{x}_t, \mathbf{u}_t) = h(F(\mathbf{x}_t, \mathbf{u}_t), \mathbf{0}) = h(\mathbf{x}_{t+1}, \mathbf{0}). \tag{1}$$

If there exist a finite number of observable functions $\{h_1, \ldots, h_m\}$ that span a subspace $\mathcal{H}$ such that $\mathcal{K}h \in \mathcal{H}$ for all $h \in \mathcal{H}$, then $\mathcal{H}$ is considered to be an invariant subspace and $\mathcal{K}$ becomes a finite-dimensional operator K.

In this work, we assume that the observables take the form $h(\mathbf{x}_t, \mathbf{u}_t) = g(\mathbf{x}_t) + L\mathbf{u}_t$, where $g(\mathbf{x}_t)$ represents an observation of state $\mathbf{x}_t$ and $L \in \mathbb{R}^{1\times p}$ is a matrix. Under the assumption of a finite-dimensional Koopman operator and defining the vector-valued observables $\mathbf{h} = [h_1, \ldots, h_m]^\intercal$ and $\mathbf{g} = [g_1, \ldots, g_m]^\intercal$, we have the update equation:

$$\begin{aligned}\mathbf{g}(\mathbf{x}_{t+1}) = K\mathbf{h}(\mathbf{x}_t, \mathbf{u}_t) &= \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} \mathbf{g}(\mathbf{x}_t) \\ \mathbf{u}_t \end{bmatrix} \\ &= A\mathbf{g}(\mathbf{x}_t) + B\mathbf{u}_t.\end{aligned} \tag{2}$$

The above expression describes the forward-time evolution of the observations $\mathbf{g}(\mathbf{x}_t)$; if A is invertible we can likewise describe the reverse-time evolution as

$$\mathbf{g}(\mathbf{x}_t) = A^{-1}\left(\mathbf{g}(\mathbf{x}_{t+1}) - B\mathbf{u}_t\right). \tag{3}$$

Consider a sequence of control inputs $\mathbf{u}_{1:T-1}$ applied to a system with a finite-dimensional Koopman operator, resulting in a sequence of states $\mathbf{x}_{1:T}$. Define the matrices $Z \in \mathbb{R}^{(m+p)\times(T-1)}$, $Y \in \mathbb{R}^{m\times(T-1)}$ as:

$$\begin{aligned} Z &= \begin{bmatrix} \mathbf{g}(\mathbf{x}_1) & \mathbf{g}(\mathbf{x}_2) & \ldots & \mathbf{g}(\mathbf{x}_{T-1}) \\ \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_{T-1} \end{bmatrix} \\ Y &= \begin{bmatrix} \mathbf{g}(\mathbf{x}_2) & \mathbf{g}(\mathbf{x}_3) & \ldots & \mathbf{g}(\mathbf{x}_T) \end{bmatrix}. \end{aligned} \tag{4}$$

Under the assumptions outlined above, we can recover A and B through $[A\ B] = YZ^\dagger$ for sufficiently long sequences, where $Z^\dagger$ is the Moore-Penrose pseudoinverse of Z. Unfortunately, the true form of the observables is generally unknown. In this work, we attempt to infer the sequence of observations $\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_T)$ based on a sequence of control inputs and the time evolution of the state of a dynamical system. Note that we do not model the functions g directly, but instead attempt to infer the value of the observations.

2.2 Inference Procedure

Consider a system subjected to a sequence of control inputs $\mathbf{u}_{1:T-1}$, causing it to traverse a set of states $\mathbf{x}_{1:T}$. We assume there exist observations of the states $\mathbf{g}(\mathbf{x}_t)$ such that the system can be simulated linearly in time as outlined in Eq. (2) and Eq. (3). Furthermore, we assume that the sequence is sufficiently long such that, when we form the matrices Z and Y as defined above, we have $[A\ B] = YZ^\dagger$, i.e. we can find the true A and B matrices directly from the observations and control inputs. Let $\mathbf{g}_t$ be a latent variable representing the observation $\mathbf{g}(\mathbf{x}_t)$. Additionally, we introduce $\mathbf{z}_{1:T}$ as a set of latent variables that enforce that multi-step predictions made with the derived dynamics model allow for accurate reconstructions of the states $\mathbf{x}_{1:T}$. Because we would expect prediction error to grow with time, we simulate the $\mathbf{z}_t$'s backward in time such that the lowest reconstruction error is generally obtained at time T, which in turn will allow more accurate predictions for how a system will evolve in future, unobserved time steps. The values of $\mathbf{z}_{1:T}$ can be determined through:

$$\mathbf{z}_T = \mathbf{g}_T, \quad \mathbf{z}_t = A^{-1}\left(\mathbf{z}_{t+1} - B\mathbf{u}_t\right). \tag{5}$$

Finally, we assume that $\mathbf{x}_t = \mathbf{g}^{-1}(\mathbf{z}_t)$, i.e. the states $\mathbf{x}_t$ are generated by inverting the observable function $\mathbf{g}(\cdot)$. Figure 1 shows the graphical model for this problem.

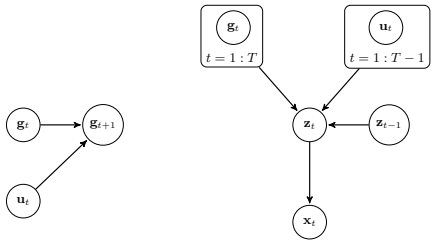

Figure 1: Graphical model. Each observation $\mathbf{g}_t$ is a function of the previous observation and control input. Because the dynamical model that governs the time evolution of $\mathbf{z}_{1:T-1}$ is derived directly from $\mathbf{g}_{1:T}$ and $\mathbf{u}_{1:T-1}$, influence flows from these variables to $\mathbf{z}_{1:T-1}$.

We desire a model that maximizes the likelihood assigned to a sequence of observed states $\mathbf{x}_{1:T}$ conditioned on actions $\mathbf{u}_{1:T-1}$. In modeling this density, we need to account for the presence of latent variables $\mathbf{g}_{1:T}$ and $\mathbf{z}_{1:T}$. We can write the expression for the likelihood $L = p(\mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1})$ in terms of the observed and latent variables as:

$$L = \int p(\mathbf{g}_{1:T}, \mathbf{z}_{1:T}, \mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1}) d\mathbf{g}_{1:T} d\mathbf{z}_{1:T}. \tag{6}$$

By the chain rule, the integrand can be factored into:

$$\begin{aligned} p(\mathbf{g}_{1:T}, \mathbf{z}_{1:T}, \mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1}) = p(\mathbf{g}_{1:T} \mid \mathbf{u}_{1:T-1}) \times \\ p(\mathbf{z}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{u}_{1:T-1})\, p(\mathbf{x}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{z}_{1:T}, \mathbf{u}_{1:T-1}) \end{aligned} \tag{7}$$

Each term can be simplified using the conditional independence assumptions encoded by the graphical model. The first term can be simplified to:

$$p(\mathbf{g}_{1:T} \mid \mathbf{u}_{1:T-1}) = p(\mathbf{g}_1) \prod_{t=2}^{T} p(\mathbf{g}_t \mid \mathbf{g}_{t-1}, \mathbf{u}_{t-1}). \tag{8}$$

Each factor in the above expression can be thought of as a (conditional) prior over an observation at time t. The second term in the integrand describes the distribution over variables $\mathbf{z}_{1:T}$, whose values can be determined exactly using Eq. (5) if $\mathbf{g}_{1:T}$ and $\mathbf{u}_{1:T-1}$ are known. Thus, we find:

$$\begin{aligned} p(\mathbf{z}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{u}_{1:T-1}) = \delta(\mathbf{z}_T \mid \mathbf{g}_T) \times \\ \prod_{t=1}^{T-1} \delta(\mathbf{z}_t \mid \mathbf{z}_{t+1}, \mathbf{g}_{1:T}, \mathbf{u}_{1:T-1}), \end{aligned} \tag{9}$$

where $\delta(\cdot \mid \cdot)$ represents a deterministic relationship. From the structure of the graphical model, the last term becomes:

$$p(\mathbf{x}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{z}_{1:T}, \mathbf{u}_{1:T-1}) = \prod_{t=1}^{T} p(\mathbf{x}_t \mid \mathbf{z}_t). \quad (10)$$

Even with these simplifications, evaluating the likelihood expression is generally intractable because it requires marginalizing over the latent variables. Therefore, instead of optimizing this objective directly, we use variational inference to optimize a lower bound. We introduce $q(\mathbf{g}_{1:T}, \mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$, an approximation of the true posterior distribution over the latent variables. Multiplying and dividing the likelihood expression by this quantity, taking the logarithm of both sides, and invoking Jensen's inequality, we find a lower bound on the log-likelihood $\ell = \log p(\mathbf{x}_{1:T} \mid \mathbf{u}_{1:T-1})$:

$$\begin{aligned}\ell \geq\, & \mathbb{E}\left[\sum_{t=1}^{T} \log p(\mathbf{x}_t \mid \mathbf{z}_t)\right] \\ & + \mathbb{E}\left[\log p(\mathbf{g}_1) + \sum_{t=2}^{T} \log p(\mathbf{g}_t \mid \mathbf{g}_{t-1}, \mathbf{u}_{t-1})\right] \\ & - \mathbb{E}\left[\log q(\mathbf{g}_{1:T}, \mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})\right],\end{aligned} \quad (11)$$

where the expectations are taken with respect to samples of $\mathbf{z}_{1:T}$ and $\mathbf{g}_{1:T}$ drawn from q.

We now consider simplified expressions for the approximate posterior distribution. The chain rule tells us that:

$$\begin{aligned}&q(\mathbf{g}_{1:T}, \mathbf{z}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = \\ &\quad q(\mathbf{g}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) q(\mathbf{z}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}).\end{aligned} \quad (12)$$

As stated previously, given knowledge of $\mathbf{g}_{1:T}$ and $\mathbf{u}_{1:T-1}$, $\mathbf{z}_{1:T}$ are known exactly. Thus, we know $\log q(\mathbf{z}_{1:T} \mid \mathbf{g}_{1:T}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = 0$. Additionally, we can factorize $q(\mathbf{g}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$ as:

$$\begin{aligned}q(\mathbf{g}_{1:T} \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) = q(\mathbf{g}_1 \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) \times \\ \prod_{t=2}^{T} q(\mathbf{g}_t \mid \mathbf{g}_{1:t-1}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}).\end{aligned} \quad (13)$$

Since all variables $\mathbf{g}_t$ are assumed to be parents of variables $\mathbf{z}_{1:T-1}$, influence can flow from all states and actions to each $\mathbf{g}_t$, and thus the above expression cannot be simplified any further based on conditional independence relationships. Taking the logarithm of the quantities in Eq. (13) and incorporating these into Eq. (11), we arrive at the following lower bound on the log-likelihood objective:

$$\begin{aligned}\ell \geq\, & \mathop{\mathbb{E}}_{\mathbf{z}_{1:T} \sim q}\left[\sum_{t=1}^{T} \log p(\mathbf{x}_t \mid \mathbf{z}_t)\right] \\ & - \mathcal{D}_{\mathrm{KL}}\left[q(\mathbf{g}_1 \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1}) \,||\, p(\mathbf{g}_1)\right] - \sum_{t=2}^{T} \mathcal{D}_{\mathrm{KL}}^{t},\end{aligned} \quad (14)$$

where $\mathcal{D}_{\mathrm{KL}}^{t}$ represents the KL-divergence between $q(\mathbf{g}_t \mid \mathbf{g}_{1:t-1}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$ and $p(\mathbf{g}_t \mid \mathbf{g}_{t-1}, \mathbf{u}_{t-1})$ for $t = 2, \ldots, T$. Thus, we have found a lower bound on our true objective that is comprised of the likelihood of the observed states $\mathbf{x}_{1:T}$ given $\mathbf{z}_{1:T}$, as well as the KL-divergence between the approximate posterior and (conditional) prior distributions over the observations $\mathbf{g}_t$. The following section provides a practical training procedure for maximizing this objective.

2.3 Optimization Procedure

The expectation in the derived lower bound can be estimated through Monte Carlo sampling. To raise the lower bound, we simultaneously optimize the parameters of six neural networks, which together comprise the Deep Variational Koopman model. The size of each neural network, which is held constant across all experiments, is listed after the name of each model (e.g. $[64, 64]$ would represent a two-layer neural network with 64 neurons in each layer).

1. The *Decoder Network* $[64, 32]$ is parameterized by $\boldsymbol{\theta}$ and outputs $\boldsymbol{\mu}_t$, the mean of a Gaussian distribution over state $\mathbf{x}_t$ given $\mathbf{z}_t$, represented by $p_{\boldsymbol{\theta}}(\mathbf{x}_t \mid \mathbf{z}_t)$. We assume that the distribution over $\mathbf{x}_t$ has constant covariance. Hence, maximizing the log-likelihood is equivalent to minimizing the square error between $\boldsymbol{\mu}_t$ and $\mathbf{x}_t$.
2. The *Temporal Encoder Network* $[64]$ is a bidirectional LSTM that maps a sequence of states $\mathbf{x}_{1:T}$ and actions $\mathbf{u}_{1:T-1}$ to a low-dimensional encoding that summarizes the system time evolution.
3. The *Initial Observation Inference Network* $[64]$ is parameterized by $\boldsymbol{\varphi}$ and outputs the parameters to a Gaussian distribution over observation $\mathbf{g}_1$ given the output of the temporal encoder, represented by $q_{\boldsymbol{\varphi}}(\mathbf{g}_1 \mid \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$.
4. The *Observation Encoder Network* $[64]$ is a recurrent neural network that takes in observations $\mathbf{g}_{1:t-1}$ and outputs an encoding describing their time evolution. The encoding is updated as more observations are sampled.
5. The *Observation Inference Network* $[64, 64]$ is parameterized by $\boldsymbol{\phi}$ and outputs the parameters to a Gaussian distribution over observation $\mathbf{g}_t$ given the output of the *Temporal* and *Observation Encoder Networks*, represented by $q_{\boldsymbol{\phi}}(\mathbf{g}_t \mid \mathbf{g}_{1:t-1}, \mathbf{x}_{1:T}, \mathbf{u}_{1:T-1})$.
6. The *Conditional Prior Network* $[64, 32]$ is parameterized by $\boldsymbol{\psi}$ and outputs the parameters to a Gaussian conditional prior distribution over observation $\mathbf{g}_t$. The output distribution is conditioned on the previous observation and action, and is represented by $p_{\boldsymbol{\psi}}(\mathbf{g}_t \mid \mathbf{g}_{t-1}, \mathbf{u}_{t-1})$.

Given a sequence of states $\mathbf{x}_{1:T}$ and actions $\mathbf{u}_{1:T-1}$, we can sample observations $\mathbf{g}_{1:T}$ from the *Observation Inference Network* and subsequently find the A and B matrices that govern the observation dynamics through $[A\ B] = YZ^{\dagger}$ and $A^{-1} = (Y - B\Gamma)X^{\dagger}$, where $\Gamma = [\mathbf{u}_1, \ldots, \mathbf{u}_{T-1}]$ and $X = [\mathbf{g}(\mathbf{x}_1), \ldots, \mathbf{g}(\mathbf{x}_{T-1})]$. Finally, $\mathbf{z}_{1:T}$ can be found through Eq. (5), and the *Decoder Network* can output state predictions.

Each time a new set of observations $\mathbf{g}_{1:T}$ is sampled, we obtain a new, globally linear dynamics model. By sampling many times, we obtain an ensemble of linear models that can provide a distribution over future outcomes for a given system. This notion of uncertainty can be appealing for a variety of tasks, including prediction and control in circumstances where data is limited. The next section details how the DVK model can enable uncertainty-aware control.

3 Control

Our goal is to select a sequence of control inputs $\mathbf{u}_{1:H}$ that minimizes $C = \sum_{t=1}^{H} c(\mathbf{x}_t, \mathbf{u}_t)$, the total incurred cost, where

$c(\mathbf{x}_t, \mathbf{u}_t)$ is the instantaneous cost. Designing controllers for nonlinear systems can be challenging, while many techniques exist for controlling linear systems. DVK models provide linear dynamics models, which makes them amenable for incorporation into many control frameworks. Below we outline considerations relating to using DVK models for control.

3.1 Cost Function

While the DVK model provides us with linear dynamics, the dynamics are linear in the latent variables $\mathbf{z}_t$. Thus, we require a cost that is a function of $\mathbf{z}_t$, not $\mathbf{x}_t$. Specifying such a cost function can be difficult; a common choice is to define the cost to be the L_2-distance between the latent representation of the current state and the representation for a goal state. Such a choice is restrictive but can be justified when the states are represented as visual inputs [Nair *et al.*, 2018].

However, it is often easier to specify the cost as a function of the state directly. For this reason, we define the cost as a function of the state and construct local quadratic approximations of the cost $\hat{c}(\mathbf{z}_t, \mathbf{u}_t)$ in the latent space. Building these local approximations requires finding the gradient and Hessian of the state cost with respect to $\mathbf{z}_t$, which in turn requires nonzero second derivatives of the activation functions in the *Decoder Network*, precluding piecewise linear ReLU activations.

3.2 Optimizing Action Sequences

We use the Differential Dynamic Programming (DDP) trajectory optimization algorithm to find an action sequence that minimizes the total predicted cost C. The DDP algorithm starts with an action sequence to form an initial reference trajectory, and uses the dynamic programming principle to iteratively update the action sequence and reference trajectory to minimize the predicted cost [Tassa *et al.*, 2012]. DDP requires locally quadratic approximations to the system dynamics and cost at all points along the reference trajectory. The DVK model provides us with linear dynamics models and the ability to find locally quadratic approximations to the cost, and thus can be readily incorporated into the DDP algorithm.

A further consideration is that some systems may have constraints on the control inputs. To account for the presence of such constraints, we optimize the action sequence $\tilde{\mathbf{u}}_{1:H}$, where we define $\mathbf{u}_t = \mathbf{u}_{\max}\tanh(\tilde{\mathbf{u}}_t)$, and $\mathbf{u}_{\max}$ represents the control saturation limits. Because of the presence of the hyperbolic tangent in this expression, the DDP algorithm requires us to make a quadratic approximation of the system dynamics with respect to the control inputs $\tilde{\mathbf{u}}_{1:H}$.

3.3 Accounting for Uncertainty

The standard DDP algorithm assumes the existence of a single, (locally) linear dynamics model. However, DVK models give us the ability to sample many possible dynamics models, which taken together can encode uncertainty about how a system will evolve in time. Accounting for such uncertainty can enable more effective control. Below are two methods we considered for performing uncertainty-aware control.

Optimize for expected cost. Given k models with $\{[A, B]_i\}_{i=1:k}$ and $\{\mathbf{z}_{1,i}\}_{i=1:k}$, construct an augmented state $\mathbf{z}_{t,\text{aug}}$ that represents the concatenation of the $\mathbf{z}_t$-values across all models. The dynamics of the augmented state will be described by creating a block-diagonal matrix out of the A-matrices and stacking the B-matrices into a single matrix. A similar procedure can be performed with the cost gradients and Hessians to find quadratic approximations to the cost function along the reference trajectory. The action sequence can be optimized according to the expected cost across all models.

Optimize for worst-case cost. Given an initial action sequence and k models, find the model predicting the largest cost and use that model to update the state and action trajectory. Subsequently find the model predicting that largest cost under the new action sequence and repeat until convergence.

3.4 Model Predictive Control

In the presence of disturbances or model errors, executing an entire action sequence determined through DDP may be inadvisable. Because DVK dynamics models will not provide perfect predictions for the time evolution of a system, we perform model predictive control (MPC) for closed-loop trajectory planning. At each time step, we feed the last T observed states and actions into the DVK model to find an ensemble of dynamics models $\{[A, B]_i\}_{i=1:k}$ and initial states $\{\mathbf{z}_{1,i}\}_{i=1:k}$. Next, we use the DDP procedures outlined above to solve for action sequence $\mathbf{u}_{1:H}$, execute the first action in the sequence, and replan at the next time step.

4 Experiments

This section evaluates the performance of the Deep Variational Koopman models on benchmark problems for dynamics modeling and control. We have limited these experiments to low-dimensional problems because it is easier to visualize whether the models have provided reasonable uncertainty estimates. However, there is no reason why DVK models could not be applied to high-dimensional systems. Future work will focus on higher-dimensional problems such as fluid flow control, as it has already been shown that Koopman-based approaches can be effective for such applications [Morton *et al.*, 2018].

4.1 Dynamics Modeling

We evaluate the ability of DVK models to learn dynamics on three benchmark problems: inverted pendulum, cartpole, and acrobot (or double pendulum). We use the OpenAI Gym [Brockman *et al.*, 2016] implementation of each environment, and modify the cartpole and acrobot environments to give them continuous action spaces.

Baseline Models

We benchmark the performance of DVK models against three baseline models. For a fair comparison, whenever a baseline shares a component with the DVK model, such as the bidirectional LSTM and decoder in the DVBF and LSTM models, we use the exact same hyperparameters across all models. In our experiments, all latent states were set to be four-dimensional.

The *Deep Variational Bayes Filter* (DVBF) model [Karl *et al.*, 2017] assumes the presence of latent states $\mathbf{z}_t$ with locally linear dynamics $\mathbf{z}_{t+1} = A_t\mathbf{z}_t + B_t\mathbf{u}_t + C_t\mathbf{w}_t$, where A_t, B_t, and C_t are functions of the current latent state and $\mathbf{w}_t$ is a noise vector. The distribution over $\mathbf{w}_1$ is output by a bidirectional

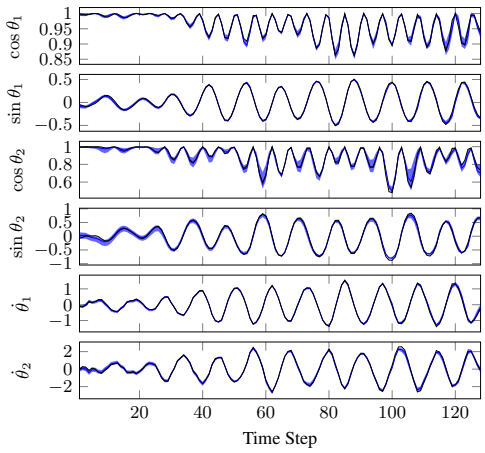

Figure 2: Results from the acrobot environment. The first 64 steps are reconstruction and the final 64 steps are prediction. Black lines indicate true state values, blue lines represent mean predictions, and shaded regions represent the range of predictions across all models.

LSTM that encodes information about the sequence of states $\mathbf{x}_{1:T}$, and $\mathbf{z}_1$ is assumed to be a function of $\mathbf{w}_1$.

The *Long Short Term Memory* (LSTM) model propagates a latent state $\mathbf{z}_t$ with a recurrent neural network and uses a decoder network to map $\mathbf{z}_t$ to $\mathbf{x}_t$. The latent state at each time step is a function of the previous latent state, the previous control input, and the network hidden state. The initial latent state $\mathbf{z}_1$ is drawn from a distribution output by a bidirectional LSTM that encodes information about states $\mathbf{x}_{1:T}$.

We train an ensemble of 10 *Multilayer Perceptrons* (MLPs). Each model is a fully-connected network trained to map the current state and action to the next state. We can use the range of predictions in the model ensemble as a measure of uncertainty. Recent work has shown that ensembles of MLPs can serve as probabilistic dynamics models that enable effective control on a variety of tasks [Chua *et al.*, 2018].

Training Details

All models were implemented in TensorFlow [Abadi *et al.*, 2015]. In each environment, 1000 trials were run for 256 time steps with random control inputs. Models were trained on subsequences of states and actions extracted from the trial data. Knowledge about a sequence of states $\mathbf{x}_{1:T}$ is required to sample the initial latent state in the DVK, DVBF, and LSTM models (and the dynamics model for DVK). For this reason, we must draw a distinction between *reconstruction*, in which a model simulates the evolution of states $\mathbf{x}_{1:T}$ about which it already has knowledge, and *prediction*, in which a model predicts the evolution of states $\mathbf{x}_{T+1:T+H}$.

In calculating $Z^\dagger$ and $X^\dagger$, a small scalar value may need to be added to the diagonal entries of Z and X to avoid conditioning issues, meaning that the A^{-1} matrix found by the DVK model might not be the true inverse of A. If the model is trained only for reconstruction, in which it simulates the system backward in time with A^{-1}, it may not perform well in prediction when it uses A for the forward-time dynamics. To ensure that the system can be simulated accurately with both A and A^{-1}, we train the DVK model to minimize the sum of the reconstruction *and* prediction errors for states $\mathbf{x}_{1:T+H}$, where we often set $T = H$. We found that employing a similar training procedure inhibited learning for the LSTM model, but did improve the predictive performance of the DVBF, and as such all DVBF results are from models trained in this manner.

Results

We evaluate the trained models on 5000 64-step test sequences from each environment. For each test sequence, we generate 10 predictions with each model. For each prediction, the DVBF and LSTM models sample different initial latent states $\mathbf{z}_1$, and the DVK model samples different values of $\mathbf{z}_T$ and dynamics matrices. We obtain distinct predictions from the MLP ensemble by generating recursive predictions with each trained model. Figure 2 provides a qualitative picture of the DVK model's ability to simulate the dynamics on one test sequence. We can observe strong agreement between the model's predictions and the true time evolution of the system, with higher uncertainty present near local minima and maxima.

The predictive performance of the models are quantified according to two metrics: (1) mean squared error (MSE) as a function of prediction horizon, averaged across the 10 predictions and 5000 trials, and (2) negative log-likelihood (NLL) of the test data as a function of prediction horizon, summed across trials. The likelihood is calculated by fitting a Gaussian distribution to the 10 predictions generated by each model and determining the probability density that distribution assigns to the true state value. Figure 3 shows model performance according to these metrics on the three studied environments. The likelihood results for the LSTM model are omitted because its fitted distributions assigned zero likelihood to some of the test data, which corresponds to infinite negative log-likelihood.

The MLP ensemble performs quite well, achieving low prediction error and assigning high likelihood to the test data. However, the results for the pendulum problem, where the prediction error grows exponentially, illustrate one drawback of using models trained to make single-step predictions. The prediction errors for such models can grow exponentially when they are used to make multi-step predictions due to errors compounding over time [Venkatraman *et al.*, 2015]. In fact, over a horizon of 128 time steps in the pendulum environment the mean-squared prediction error for the MLP ensemble grows to values on the order of 10^6. The DVK model achieves competitive performance with the MLP ensemble, while not suffering from the same instabilities and also providing linear dynamics models that can be used more easily for control.

The DVBF outperforms the LSTM baseline, and attains performance that is often close to that of the DVK model, but at a much higher computational cost. The DVK model computes a single dynamics model that it uses to propagate the latent state for all time steps, while the DVBF must compute a new dynamics model, which is a nonlinear function of the current latent state, at each time step. Therefore, the computational graph for the DVBF takes significantly longer to compile, and furthermore in our experiments the time required to perform a forward and backward pass during training was approximately an order or magnitude longer for the DVBF.

4.2 Control

We evaluate the effectiveness of the control procedure detailed in Section 3 on the inverted pendulum environment, in which

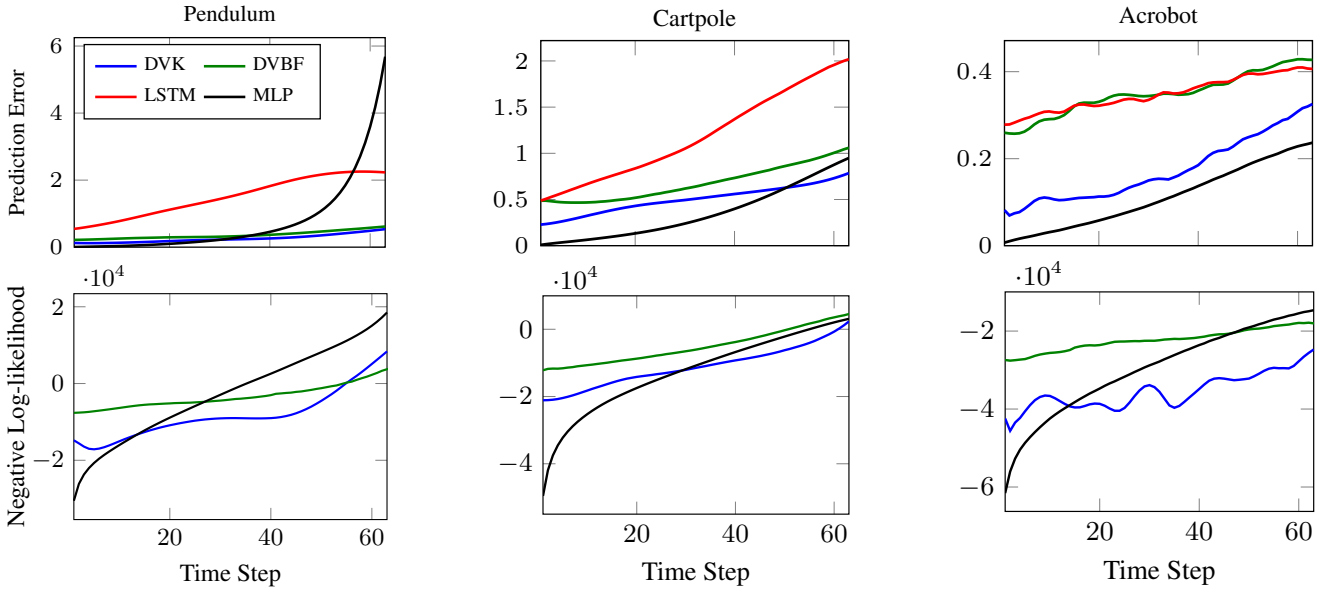

Figure 3: Top: MSE as a function of prediction horizon. Bottom: NLL assigned to the test data (lower is better).

Number of Models	Cost		Fraction of Time Vertical		Falls per Trial	
	Worst Case	Expected	Worst Case	Expected	Worst Case	Expected
1	-	370.7	-	0.632	-	0.353
5	307.5	285.7	0.710	0.732	0.312	0.111
10	291.9	274.3	0.733	0.748	0.218	0.063
30	**251.6**	270.9	**0.771**	0.748	0.061	**0.055**

Table 1: Control Performance

the goal is to swing up and balance an underactuated pendulum. The cost function penalizes deviations of the pendulum from vertical, as well as nonzero angular velocities and control inputs. In each episode, the pendulum is initialized in a random state and the system is simulated for 256 time steps.

We ran 50 trials with random control inputs and trained a DVK model on the trial data with a reconstruction and prediction horizon of $T = H = 16$. Because the original dataset did not contain many instances where the pendulum was near the goal state, we ran 20 additional trials where the actions were selected through MPC optimizing for expected cost with five sampled models. We then finetuned the DVK model on data from these trials before carrying out the final experiments.

The results for different ensemble sizes and optimization procedures, taken from 1000 seeded trials, can be found in Table 1. We quantify performance according to (1) the average cost incurred in each trial, (2) the fraction of the time the pendulum was vertical ($\theta \in [-\pi/8, \pi/8]$) across all trials, and (3) the average number of falls per trial. A fall is classified as a scenario where $\theta \notin [-\pi/8, \pi/8]$ after being in the interval for more than 20 time steps. The best performance according to each metric is highlighted in bold.

The results show a clear benefit from sampling more models. The trend in performance improvement is more pronounced for the worst-case optimization scheme; when optimizing for expected cost we do see a benefit to sampling more models, but with diminishing returns. The best performance is obtained with 30 models, with the lowest average cost achieved through a worst-case optimization procedure. However, optimizing for expected cost leads to much better performance for smaller model ensembles, and thus could be a preferable approach if the goal is to obtain satisfactory performance while keeping the number of sampled models low.

5 Conclusions

We introduced the Deep Variational Koopman model, a method for inferring Koopman observations and sampling ensembles of linear dynamics models that can be used for prediction and control. We demonstrated that DVK models were able to perform accurate, long-term prediction on a series of benchmark tasks, and that accounting for the uncertainty encoded by multiple sampled models improved controller performance on the inverted pendulum task. Future work will focus on applying the DVK models to higher-dimensional problems and more complex tasks, such as fluid flow control. Source code associated with this project can be found at https://github.com/sisl/variational_koopman.

Acknowledgments

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE- 114747 and the Stanford Energy Alliance. The authors would like to thank Zac Manchester for valuable feedback.

References

[Abadi *et al.*, 2015] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems. *arXiv preprint arXiv:1603.04467*, 2015.

[Banijamali *et al.*, 2018] Ershad Banijamali, Rui Shu, Mohammad Ghavamzadeh, Hung Hai Bui, and Ali Ghodsi. Robust locally-linear controllable embedding. In *International Conference on Artificial Intelligence and Statistics, (AISTATS)*, 2018.

[Brockman *et al.*, 2016] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[Chua *et al.*, 2018] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[Fraccaro *et al.*, 2017] Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[Hafner *et al.*, 2018] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[Kaiser *et al.*, 2017] Eurika Kaiser, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of Koopman eigenfunctions for control. *arXiv preprint arXiv:1707.01146*, 2017.

[Karl *et al.*, 2017] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational Bayes filters: Unsupervised learning of state space models from raw data. In *International Conference on Learning Representations (ICLR)*, 2017.

[Koopman, 1931] B. O. Koopman. Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[Korda and Mezić, 2018] Milan Korda and Igor Mezić. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149–160, 2018.

[Krishnan *et al.*, 2017] Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *AAAI Conference on Artificial Intelligence*, 2017.

[Li *et al.*, 2017] Qianxiao Li, Felix Dietrich, Erik M Bollt, and Ioannis G Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the Koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[Lusch *et al.*, 2018] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):4950, 2018.

[Moerland *et al.*, 2017] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Learning multimodal transition dynamics for model-based reinforcement learning. *arXiv preprint arXiv:1705.00470*, 2017.

[Morton *et al.*, 2018] Jeremy Morton, Freddie D Witherden, Antony Jameson, and Mykel J Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[Nair *et al.*, 2018] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[Otto and Rowley, 2017] Samuel E Otto and Clarence W Rowley. Linearly-recurrent autoencoder networks for learning dynamics. *arXiv preprint arXiv:1712.01378*, 2017.

[Proctor *et al.*, 2018] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Generalizing Koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

[Rangapuram *et al.*, 2018] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[Takeishi *et al.*, 2017] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning Koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[Tassa *et al.*, 2012] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[Venkatraman *et al.*, 2015] Arun Venkatraman, Martial Hebert, and J. Andrew Bagnell. Improving multi-step prediction of learned time series models. In *AAAI Conference on Artificial Intelligence*, 2015.

[Watter *et al.*, 2015] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.