# Modeling and Planning under Uncertainty using Deep Neural Networks

Daniel L. Marino and Milos Manic *Senior Member, IEEE*

***Abstract*—Artificial Neural Networks (ANNs) have been frequently used in industrial applications to model complex systems. However, using traditional ANNs for long-term planning tasks remains a challenge as they lack the capability to model uncertainty. Process noise and approximation errors cause ANN long-term estimations to deviate from the real behavior of the system. Unlike traditional ANNs, stochastic models provide a natural way to model uncertainty, providing estimations over a range of several possible outcomes. This paper introduces a stochastic modeling and planning approach using Deep Bayesian Neural Networks (DBNNs). We use DBNNs to learn a stochastic model of the system dynamics. Planning is addressed as an open-loop trajectory optimization problem. We present two approaches for learning the dynamics: using single-step predictions and using multi-step predictions. The advantages of the proposed methodology are: 1) accurate long-term estimations of the system state-trajectory probability distribution without the need for expert knowledge of the dynamics; 2) improved generalization and faster convergence rates in the trajectory optimization task when using multi-step predictions to train the model; 3) viable for real-world applications since all expensive optimizations are executed offline while using a reasonable number of data-samples. Testing is performed using challenging underactuated benchmark problems: the Cartpole and Acrobot. The presented methodology successfully learned the swing-up maneuver using a relatively small number of iterations, with less than 125 sampled trajectories, and without any expert knowledge of the dynamics.**

***Index Terms*—Trajectory optimization, uncertainty, Bayesian neural-networks, variational inference.**

## I. Introduction

Control of underactuated non-linear systems is often broken down into three tasks [1] [2]: modeling system dynamics, open-loop trajectory planning, and trajectory stabilization. Historically, modeling has been performed using first principles. However, this approach is labor intensive and requires highly skilled professionals. A promising alternative is to use data-driven models to learn the system dynamics. Fully data-driven models provide a flexible approach that does not require the derivation of complex mathematical models from first principles, reducing the need for expert knowledge [3].

Deterministic neural-networks have been extensively used in the control of industrial systems, some applications include: controlling electric drives [4] [5], robot manipulators [6] and HVAC systems [7].

Despite the universal approximation properties of ANNs, standard deterministic ANNs lack the capacity of modeling uncertainty. As a consequence, deterministic ANNs are sensitive to model-mismatch problems where long-term estimations deviate from the behavior of the real system [3] [8]. Because of the inherent stochastic nature of real systems and the approximation errors introduced by data-driven modeling, we cannot expect an algorithm to provide long-term predictions with 100% accuracy. A more realistic approach is to design a model that provides accurate long-term predictions of the probability distribution of the system state trajectory. The estimations should take into account the uncertainties of both the plant and the learned model.

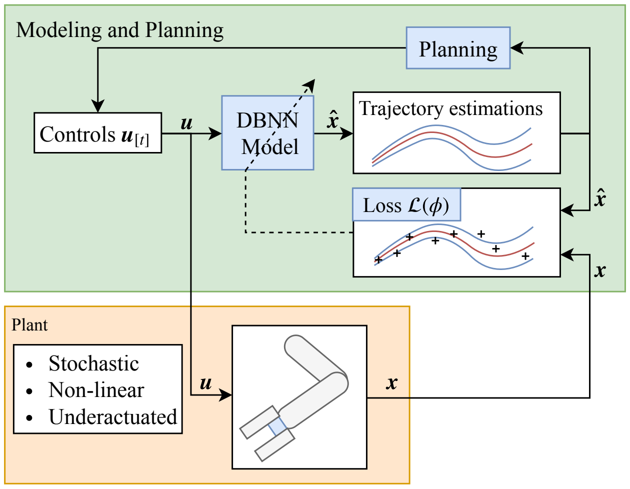


Fig. 1. Overview: Modeling and planning using DBNNs.

Stochastic data-driven models provide a natural way to model uncertainty, which can be used to mitigate model-mismatch problems [3]. These models allow us to make informed decisions using the predictions of the model while being cautious about the uncertainty of such predictions [9].

In this paper, we address the first two stages of controlling underactuated systems: modeling system dynamics and open-loop trajectory planning. Our methodology uses a Deep Bayesian Neural Network (DBNN) to learn a stochastic model of the system dynamics. The learned model is used to estimate the probability distribution of long-term system state trajectories. Planning is addressed as an open-loop trajectory optimization problem, where long-term trajectory estimations are used to find optimal control inputs that accomplish a given task. Fig. 1 shows an overview of the methodology.

The *Contributions* of the paper are: 1) a methodology for open-loop planning using DBNN stochastic models, we focus on studying the performance of long-term trajectory estimations which are essential for open-loop control; 2) two approaches for learning system dynamics: using single-step

Daniel L. Marino and Milos Manic are with the Department of Computer Science, Virginia Commonwealth University, Richmond, VA 23284 USA (e-mail: marinodl@vcu.edu; misko@ieee.org).

Accepted version of the paper appearing in the IEEE Transactions on Industrial Informatics. DOI: 10.1109/TII.2019.2917520

predictions and using multi-step predictions; 3) a set of tools to achieve stable multi-step training, especially when using heteroscedastic models; 4) a comparative analysis between homoscedastic and heteroscedastic models. For experimental evaluation we used the Cartpole and Acrobot benchmark problems.

In this paper, we mainly focus on uncertainties derived from the approximation nature of the learned model. Uncertainties in this paper arise from: 1) unexplored areas where data is not available; 2) approximation errors of the DBNN model; 3) time discretization; 4) uncertainty from numerical computations in the simulation.

The rest of the paper is organized as follows: Section II describes the related work on the area of data-driven modeling and control. Section III provides a background of DBNNs and its training using variational inference. Section IV describes the trajectory optimization task and how the system dynamics are learned using single-step and multi-step predictions. Section V presents the experimental evaluation. Section VI concludes the paper.

## II. Related work

Given the success of Deep Learning in high dimensional problems[10] and reinforcement learning [11] [12], there is an increasing interest in applying Deep Neural Networks (DNNs) in industrial applications. Forecasting [13] [14], fault diagnosis [15] [16] [17], and continuous low-level control [18] [19] [8] are some of the recent applications of DNNs.

There is also a growing interest in learning stochastic models for reinforcement learning and optimal control. In [20] Variational Bayes is used for robust identification of industrial processes. In [3] the PILCO algorithm is introduced. The algorithm uses Gaussian processes to model system dynamics and provide uncertainty estimations. In [21] Gaussian process state-space models are trained using variational inference, providing a mechanism to trade off model capacity and computation time. In [22] the PILCO algorithm is modified to use DBNNs, alleviating the problems when working with high-dimensional large datasets. In [23], a combination of bootstrapping and dropout is used to estimate uncertainties on collision avoidance tasks. In [24] the standard Gaussian process model is extended to handle sequential data by using an LSTM model. Recent work has been focused on extending the modeling capabilities of DBNN architectures to include heteroscedastic and multi-modal distributions [25] [26].

Most control applications of ANNs and DBNNs found in literature focus on closed loop-control with some variation of actor-critic design, model predictive control (MPC) or back-propagation of the dynamics through a feedback controller. The presented approach is based on the methodology presented in [22] where DBNN are used for feedback control. In contrast to the methodologies presented by [3] [22] [25] which focus on feedback control, in this paper we are interested in evaluating the viability of using DBNNs for open-loop trajectory optimization. In contrast to MPC approaches, our approach executes all optimizations offline. Once the optimal trajectory is computed, our approach executes the trajectory in open-loop, as opposed to MPC where the optimization is executed in real-time.

## III. Stochastic modeling using Bayesian neural-networks

This section presents a brief overview of training DBNNs using variational inference [27] [28]. Given a dataset $\mathcal{D} = \left\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) | \boldsymbol{y}^{(i)} \in \boldsymbol{Y}, \boldsymbol{x}^{(i)} \in \boldsymbol{X}\right\}_{i=1}^{|\mathcal{D}|}$ of input/output pairs of samples $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, we would like to learn a probability distribution $p(\boldsymbol{y}|\boldsymbol{x}, w)$, parameterized by $w$, that would allow us to make predictions over previously unseen test points $\boldsymbol{y}^*, \boldsymbol{x}^*$.

In the Bayesian setting, the probability distribution for a test point $\boldsymbol{x}^*$ can be expressed as follows:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*, \boldsymbol{Y}, \boldsymbol{X}) = \int p(\boldsymbol{y}^*|\boldsymbol{x}^*, w) p(w|\boldsymbol{Y}, \boldsymbol{X}) dw \tag{1}$$

where $p(w|\boldsymbol{Y}, \boldsymbol{X})$ is the probability of $w$ given the training dataset $\boldsymbol{Y}, \boldsymbol{X}$. In other words, $p(w|\boldsymbol{Y}, \boldsymbol{X})$ quantifies how well a given value of $w$ represents the data sampled from the real system. Using Bayes theorem, this distribution is defined as follows:

$$p(w|\boldsymbol{Y}, \boldsymbol{X}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X}, w) p(w)}{p(\boldsymbol{Y}|\boldsymbol{X})}$$

where $p(w)$ is the prior probability for the parameters of the model. In practice, the prior is used to prevent over-fitting.

As $p(w|\boldsymbol{Y}, \boldsymbol{X})$ is typically intractable, an approximation $q_\phi(w)$ is used in practice [28]. The Kullback-Leibler (KL) divergence is used to measure the difference between these distributions. The objective then becomes to find the parameters $\phi^*$ of the distribution $q_\phi(w)$ that minimize the difference between the distributions $p(w|\boldsymbol{Y}, \boldsymbol{X})$ and $q_\phi(w)$:

$$\phi^* = \arg\min_\phi KL\left(q_\phi(w) \| p(w|X, Y)\right) \tag{2}$$

By solving the minimization problem in Eq. (2), we find the distribution $q_{\phi^*}(w)$ that best approximates $p(w|\boldsymbol{Y}, \boldsymbol{X})$. From the definition of the KL divergence, we can obtain a more convenient way to express Eq. (2):

$$\begin{aligned} KL\left(q_\phi(w) | p(w|\boldsymbol{X}, \boldsymbol{Y})\right) &= \int_w q_\phi(w) \ln\left(\frac{q_\phi(w)}{p(w|\boldsymbol{X}, \boldsymbol{Y})}\right) dw \\ &= \underbrace{\left[\mathop{\mathbb{E}}_{w \sim q_\phi(w)} \left[-\ln\left(p(\boldsymbol{Y}|\boldsymbol{X}, w)\right)\right] + KL\left(q_\phi(w) \| p(w)\right)\right]}_{\text{negative log evidence lower bound } \mathcal{L}(\phi)} \\ &\quad + \ln\left(p(\boldsymbol{Y}|\boldsymbol{X})\right) \end{aligned}$$

Given that $P\left(\boldsymbol{Y}|\boldsymbol{X}\right)$ is constant (does not depend on $\phi$), minimizing the negative *log evidence lower bound* (ELBO) $\mathcal{L}\left(\phi\right)$ is equivalent to minimizing the KL divergence in Eq. (2) [27][28].

Assuming our dataset $\mathcal{D}$ is composed of $|\mathcal{D}|$ number of independent and identically distributed samples the loss w.r.t. the variational parameters $\phi$ can be expressed as follows:

$$\mathcal{L}(\phi) = \sum_{i=1}^{|\mathcal{D}|} \mathop{\mathbb{E}}_{q_\phi(w)} \left[-\ln\left(p(\boldsymbol{y}^{(i)}|\boldsymbol{x}^{(i)}, w)\right)\right] + KL\left(q_\phi(w) \| p(w)\right) \tag{3}$$

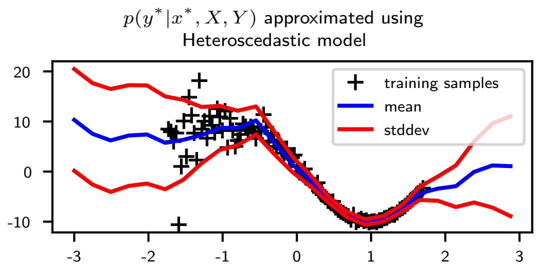


Fig. 2. Heteroscedastic distribution learned using a Bayesian neural-network. The estimated standard deviation shows how the uncertainty grows as we move away from the training dataset

In sections IV-A and IV-B, $\mathcal{L}(\phi)$ will be used as the loss for learning systems dynamics, quantifying how well the model approximates the data sampled from the real system. The loss $\mathcal{L}(\phi)$ is composed of two main terms: $\mathbb{E}_{w\sim q_\phi(w)}\left[\ln\left(p(\boldsymbol{Y}|\boldsymbol{X},w)\right)\right]$ measures how well the data-driven model fits the data; $KL\left(q_\phi(w)\|p(w)\right)$ usually works as a regularizer, and often takes the form of an $l_2$ regularization of the neural-network weights (See Appendix A).

We can optimize Eq. (3) using a Monte-Carlo approach to approximate the expectation. Furthermore, back-propagation can be used to calculate the gradients w.r.t. $\phi$ if $q_\phi(w)$ is represented using the re-parameterization trick [29]. Dropout is an example of a simple way to re-parameterize a variational distribution $q_\phi(w)$ that takes the form of a Mixture of Gaussians[28].

A common distribution adopted for $p(\boldsymbol{y}|\boldsymbol{x},w)$ is a multivariate Gaussian distribution with diagonal precision $\hat{\boldsymbol{\tau}}$:

$$p(\boldsymbol{y}|\boldsymbol{x},w) = \mathcal{N}\left(\boldsymbol{y}\mid\hat{\boldsymbol{\mu}}(\boldsymbol{x},w_\mu),\ \text{diag}\left(\hat{\boldsymbol{\tau}}(\boldsymbol{x},w_\tau)\right)^{-1}\right) \tag{4}$$

where $\hat{\boldsymbol{\mu}}(\boldsymbol{x},w_\mu), \hat{\boldsymbol{\tau}}(\boldsymbol{x},w_\tau)$ are functions parameterized by $w = w_\mu \cup w_\tau$. $\hat{\boldsymbol{\mu}}$ approximates the mean of the Gaussian distribution, while $\hat{\boldsymbol{\tau}}$ approximates the reciprocal of the variance.

Using Eq. (4) as our model, the loss in Eq. (3) can be estimated as follows:

$$\mathcal{L}(\phi) \approx \frac{1}{2\,|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}\mathcal{L}_R\left(\boldsymbol{y}^{(i)},\boldsymbol{x}^{(i)}\right) + \frac{1}{|\mathcal{D}|}KL\left(q_\phi(w)\|p(w)\right) \tag{5}$$

where

$$\begin{aligned}\mathcal{L}_R\left(\boldsymbol{y},\boldsymbol{x}\right) &= -\log(\hat{\boldsymbol{\tau}}_{\boldsymbol{x}})^T\mathbf{1} + \left\|\sqrt{\hat{\boldsymbol{\tau}}_{\boldsymbol{x}}}\odot\left(\boldsymbol{y}-\hat{\boldsymbol{\mu}}_{\boldsymbol{x}}\right)\right\|^2\\ \hat{\boldsymbol{\mu}}_{\boldsymbol{x}} &= \hat{\boldsymbol{\mu}}(\boldsymbol{x},w_\mu); \quad w\sim q_\phi(w)\\ \hat{\boldsymbol{\tau}}_{\boldsymbol{x}} &= \hat{\boldsymbol{\tau}}(\boldsymbol{x},w_\tau)\end{aligned}$$

We use $\odot$ to denote the element-wise (Hadamard) product. $\mathbf{1}$ represents a vector filled with ones. Note that Eq. (5) is a Monte-Carlo approximation of the loss, using parameters $w$ sampled from the variational distribution $w\sim q_\phi(w)$.

Having obtained the parameters $\phi^*$ that minimize Eq. (5), the distribution $q_{\phi^*}(w)$ can be used to estimate Eq. (1) (the test prediction) using:

$$p(\boldsymbol{y}^*|\boldsymbol{x}^*,\boldsymbol{X},\boldsymbol{Y}) \approx \mathop{\mathbb{E}}_{w\sim q_{\phi^*}(w)}\left[p(\boldsymbol{y}^*|\boldsymbol{x}^*,w)\right]$$

Depending on how $\hat{\boldsymbol{\tau}}$ is defined, we can use Eq. (4) to model homoscedastic and heteroscedastic distributions. Homoscedastic models assume the variance is constant, while heteroscedastic models assume the variance changes depending on the inputs $\boldsymbol{x}$. Fig. 2 shows an example of an heteroscedastic distributions modeled using DBNNs. This paper makes emphasis on heteroscedastic models, as they are more general, challenging and commonly found in industrial systems [30].

In this paper, $\hat{\mu}(\boldsymbol{x},w_\mu)$ is modeled using a multi-layer perceptron with dropout. For homoscedastic models, the precision is defined as a constant $\hat{\tau}(\boldsymbol{x},w_\tau) = \boldsymbol{\tau}_w^2$, which is jointly optimized when minimizing Eq. (5). For heteroscedastic models, $\hat{\tau}(\boldsymbol{x},w_\mu)$ is a multi-layer perceptron without dropout. Appendix A describes in detail the variational distributions and priors used to model the parameters.

## IV. Modeling and planning using DBNNs

This section describes the presented approach for modeling and planning under uncertainty using Deep Bayesian Neural-Networks (DBNNs). Specifically, we present how DBNNs are trained to approximate non-linear dynamic systems using single-step and multiple-step predictions. Planning is formulated as a trajectory optimization problem, where controls are applied in open-loop.

In this paper, we consider fully observable dynamic systems. We assume a stochastic dynamic system model that takes the following form:

$$\begin{aligned}\boldsymbol{z}_{[t+1]} &= \boldsymbol{z}_{[t]} + \Delta_{[t]} && (6)\\ \Delta_{[t]} &\sim \mathcal{N}\left(\boldsymbol{\mu}_{[t]}, \text{diag}\left(\boldsymbol{\tau}_{[t]}\right)^{-1}\right) && (7)\\ \boldsymbol{\mu}_{[t]} &= \boldsymbol{\mu}\left(\boldsymbol{z}_{[t]}\oplus\boldsymbol{u}_{[t]}, w_\mu\right)\\ \boldsymbol{\tau}_{[t]} &= \boldsymbol{\tau}\left(\boldsymbol{z}_{[t]}\oplus\boldsymbol{u}_{[t]}, w_\tau\right)\end{aligned}$$

where $\boldsymbol{z}_{[t]}$ is the state of the system at time step $t$, and $\Delta_{[t]}$ is the increment when the control signal $\boldsymbol{u}_{[t]}$ is applied on the system. The symbol $\oplus$ represents the concatenation operator. $\Delta_{[t]}$ is modeled using the DBNN described in Eq. (4), with inputs/outputs defined as $\boldsymbol{x} = \boldsymbol{z}\oplus\boldsymbol{u}$ and $\boldsymbol{y} = \Delta$. Fig. 3a shows the architecture of the DBNN.

Modeling $\Delta_{[t]}$ using a Gaussian distribution serves the purpose of explicitly including a mechanism to handle unmodeled disturbances (process noise). The variational posterior placed on the parameters serves the purpose of dealing with the uncertainties from the learned model, increasing uncertainty in areas where data is not available. In this paper, we assume zero measurement noise. Note that in Eq. (6) the noise derived from $\Delta_{[t]}$ is propagated over time. This formulation ensures that uncertainty increases over time.

The trajectory optimization problem in this paper is defined as a finite-horizon open-loop control problem with deterministic control inputs $\boldsymbol{u}_{[1]},...,\boldsymbol{u}_{[T_c]}$. Finite-horizon problems look for a set of optimal inputs $\boldsymbol{u}_{[t]}$ that minimize a control cost that depends on the state of the system for a finite number of time steps. The control cost quantifies how effective the inputs are in driving the system towards completing a particular task. In this paper, we only consider open-loop control, which means that the control signal $\boldsymbol{u}_{[t]}$ does not depend on the current state $\boldsymbol{z}_{[t]}$. In contrast, closed-loop approaches look for control policies that depend on the current state of the system (i.e. $\boldsymbol{u}_{[t]} = g(\boldsymbol{z}_{[t]})$).

Concretely, the objective of the trajectory optimization problem is to find the control inputs $\boldsymbol{u}_{[t]}$ that minimize the control cost $\mathcal{L}_c$:

$$\min_{\{\boldsymbol{u}_{[1]},\dots,\boldsymbol{u}_{[T_c]}\}} \mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right)$$

where $\boldsymbol{z}_{[t]}$ is the state of the system at time step $t$, $\boldsymbol{z}_{[1]}$ is the initial state of the system, and the control cost $\mathcal{L}_c$ is defined as follows:

$$\mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right) = \underset{\boldsymbol{z}_{[T_c]}}{\mathbb{E}}\left[\mathcal{L}_{T_c}\left(\boldsymbol{z}_{[T_c]}, \boldsymbol{u}_{[T_c]}\right)\right] + \sum_{t=1}^{T_c-1} \underset{\boldsymbol{z}_{[t]}}{\mathbb{E}}\left[\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right)\right] \quad (8)$$

**Algorithm 1** Trajectory Optimization using DBNN models

**Input:** Initial state $\boldsymbol{z}_{[1]}$, control penalty $\mathcal{L}_c(\cdot)$

**Output:** Sequence of optimal control inputs $\left\{\boldsymbol{u}^*_{[t]}\right\}_{t=1}^{T_c}$

```
1: Initialize dataset $\mathcal{D}$ with $U_o$ system trajectories of length
   $T_c$, using random control inputs
   $\mathcal{D} \leftarrow \left\{(\boldsymbol{\Delta}^{(i)}_{[t]}, \boldsymbol{z}^{(i)}_{[t]}, \boldsymbol{u}^{(i)}_{[t]})\right\}_{i,t=1}^{i=U_o,t=T_c}$
2: for $j = 1$ to max-iter do
3:     Modeling: Fit dynamics by minimizing:
           $\phi^* = \arg\min \mathcal{L}(\phi)$
       where $\mathcal{L}(\phi)$ is either:
       • The single-step loss in Eq. (9) (section IV-A)
       • The multi-step loss in Eq. (10) (section IV-B)
4:     Planning: Run trajectory optimization by minimizing
       the control cost in Eq. (8):
           $\boldsymbol{u}^*_{[t]} = \arg\min \mathcal{L}_c\left(\boldsymbol{z}_{[1]}, \boldsymbol{u}_{[1]}, ..., \boldsymbol{u}_{[T_c]}\right)$
       Section IV-C describes how the expectations on Eq. (8)
       are approximated using particles.
5:     Collect $U$ new trajectories from the real system using
       $\boldsymbol{u}_{[t]} = \boldsymbol{u}^*_{[t]} + \lambda\boldsymbol{\epsilon}$, where $\lambda$ is used as an exploration
       parameter
       $\mathcal{D} \leftarrow \mathcal{D} \cup \left\{(\boldsymbol{\Delta}^{(i)}_{[t]}, \boldsymbol{z}^{(i)}_{[t]}, \boldsymbol{u}^{(i)}_{[t]})\right\}_{i,t=1}^{i=U,t=T_c}$
6: end for
7: return Sequence of optimal controls $\boldsymbol{u}^*_{[t]}$
```

Algorithm 1 describes the open-loop trajectory optimization approach using DBNN models. Planning is performed completely offline and depends only on the control loss and the long-term estimations of the DBNN model. We used ADAM [31] for all optimization tasks. In each iteration, new data-samples are collected from the system in order to to continuously improve the learned DBNN model. Fig. 3b and 3c show the flow-chart for single-step and multi-step training, respectively. Single-step uses one-step prediction for learning the system dynamics. Multi-step uses predictions over several time steps. The following sections describe the details of single-step and multi-step approaches for fitting the dynamics.

### A. Training using single-step prediction

Given the Markovian properties of fully-observable systems, a common approach to fit the dynamics is by using single-step predictions [3] [22]. Under this approach, the training loss fits the dynamics by only taking into account predictions made one step ahead. For a mini-batch $\left\{(\boldsymbol{\Delta}^{(i)}, \boldsymbol{z}^{(i)}, \boldsymbol{u}^{(i)})\right\}$ of $|\boldsymbol{X}|$ number of samples the loss is expressed as follows:

$$\mathcal{L}(\phi) = \frac{1}{2\,|\boldsymbol{X}|}\sum_{i=1}^{|\boldsymbol{X}|}\mathcal{L}_R\left(\boldsymbol{\Delta}^{(i)}, \boldsymbol{z}^{(i)} \oplus \boldsymbol{u}^{(i)}\right) + \frac{1}{|\mathcal{D}|}KL\left(q_\phi(w)\|p(w)\right) \quad (9)$$

where $|\mathcal{D}|$ is the total number of samples in the training dataset. Note that the single-step loss in Eq. (9) is defined w.r.t. the parameters $\phi$ of the variational distribution $q_\phi(w)$. As shown in Fig. 3a, Dropout is only applied to $\hat{\boldsymbol{\mu}}$.

### B. Training using multi-step predictions

Improved performance has been observed when fitting sequential models using multi-step/long-term predictions [14]. By forcing the model to use its own predictions to predict further in the future, we ensure that the model takes into account the compounding errors that get propagated over time, thus improving generalization and encouraging learning better internal representations.

Multi-step training is performed using mini-batches of size $|\boldsymbol{X}|$, composed of an initial state $\boldsymbol{z}^{(i)}_{[1]}$ with a sequence of control and state increments:

$$\text{mini-batch} = \left\{\boldsymbol{z}^{(i)}_{[1]}, \left\{\left(\boldsymbol{\Delta}^{(i)}_{[t]}, \boldsymbol{u}^{(i)}_{[t]}\right)\right\}_{t=1}^{T_m}\right\}_{i=1}^{|\boldsymbol{X}|}$$

where $\boldsymbol{u}^{(i)}_{[t]}$ is the control input of sample $i$ at time $t$ and $\boldsymbol{\Delta}^{(i)}_{(t)}$ is the corresponding state increment. Training using multi-step predictions is achieved using the following loss:

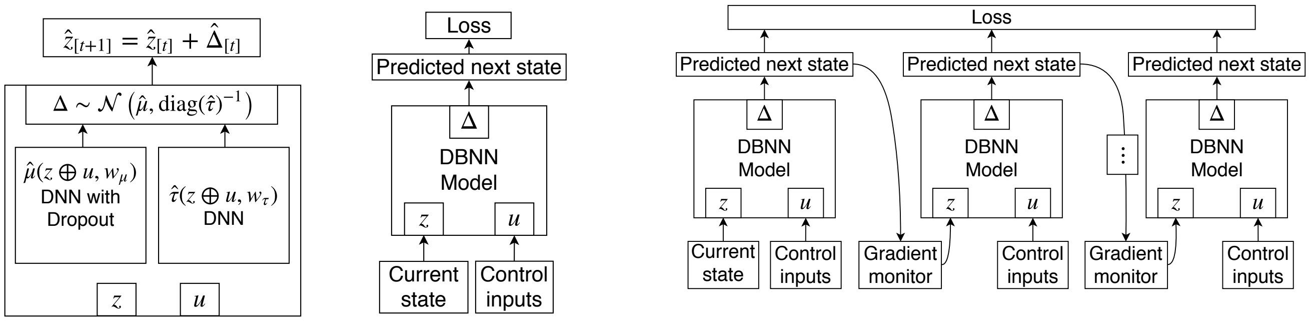
$$\mathcal{L}(\phi) = \frac{1}{2\,|\boldsymbol{X}|\,T_m}\sum_{i,t=1}^{|\boldsymbol{X}|,T_m}\mathcal{L}_R\left(\boldsymbol{\Delta}^{(i)}_{[t]}, \hat{\boldsymbol{z}}^{(i)}_{[t]} \oplus \boldsymbol{u}^{(i)}_{[t]}\right) + \frac{1}{|\mathcal{D}|}KL\left(q_\phi(w)\|p(w)\right) \quad (10)$$

where $\hat{\boldsymbol{z}}^{(i)}_{[t]}$ is the estimated state at time $t$:

$$\hat{\boldsymbol{z}}^{(i)}_{[t]} = \boldsymbol{z}^{(i)}_{[1]} + \sum_{k=1}^{t-1}\hat{\boldsymbol{\mu}}^{(i)}_{[k]} \odot \boldsymbol{\epsilon}\frac{1}{\sqrt{\hat{\boldsymbol{\tau}}^{(i)}_{[k]}}} \quad (11)$$

where:

$$\hat{\boldsymbol{\mu}}^{(i)}_{[k]} = \hat{\boldsymbol{\mu}}\left(\hat{\boldsymbol{z}}^{(i)}_{[k]} \oplus \boldsymbol{u}^{(i)}_{[k]}, w_\mu\right), \quad w_\mu \sim q_\phi(w)$$
$$\hat{\boldsymbol{\tau}}^{(i)}_{[k]} = \hat{\boldsymbol{\tau}}\left(\hat{\boldsymbol{z}}^{(i)}_{[k]} \oplus \boldsymbol{u}^{(i)}_{[k]}, w_\tau\right)$$



$\hat{z}_{[t+1]} = \hat{z}_{[t]} + \hat{\Delta}_{[t]}$
$\Delta \sim \mathcal{N}\left(\hat{\mu}, \mathrm{diag}(\hat{\tau})^{-1}\right)$
$\hat{\mu}(z \oplus u, w_\mu)$ DNN with Dropout
$\hat{\tau}(z \oplus u, w_\tau)$ DNN
z
u
Loss
Predicted next state
Δ
DBNN Model
Current state
Control inputs
Gradient monitor

(a) DBNN model (b) Single-step training (c) Multi-step training

Fig. 3. Modeling dynamic systems using DBNNs

The loss in Eq. (10) is a standard unrolling of the neural-network, usually employed in the Backpropagation Through-Time algorithm [32].

The multi-step propagation in Eq. (11) uses a single sample realization of the DBNN probability distribution. This single sample realization is used in the multi-step loss from Eq. (10), which can be interpreted as a single-sample MC approximation of the expectation in Eq. (3). This method provides a noisy but efficient way to perform mini-batch optimization for learning the dynamics of the system. A more accurate estimation of the multi-step loss can be made by using multiple-particles (see section IV-C). However, using multiple particles leads to excessive computational burden. In the experiments we evidenced that using a single particle is enough to obtain a stable optimization for learning the system dynamics.

One of the challenges posed by multi-step training is unstable gradient propagation. Back-propagation is well known to suffer from exploding/vanishing gradient problems when applied over unrolled networks [33]. Furthermore, compounding errors that result from model mismatch can also destabilize the optimization.

In order to ensure a stable propagation of gradients, we took the following actions: 1) weights are initialized according to [34]; 2) impose the bounds $(\boldsymbol{\tau}_{\min}, \boldsymbol{\tau}_{\max})$ on $\hat{\tau}$; 3) monitor the norm of the gradients.

Heteroscedastic models are particularly sensitive to unstable gradient propagation when trained using multi-step predictions. The instability can be understood by looking at the gradients of $\mathcal{L}_R$ w.r.t. $\hat{\boldsymbol{\tau}}$ and $\hat{\boldsymbol{\mu}}$ (see Appendix B for derivation):

$$\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\tau}}} = -\frac{1}{\hat{\boldsymbol{\tau}}} + (\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}})^2 \tag{12}$$

$$\frac{\partial \mathcal{L}_R}{\partial \hat{\boldsymbol{\mu}}} = -2\hat{\boldsymbol{\tau}} \odot (\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}}) \tag{13}$$

In order to ensure stable gradient propagation, we need to ensure the value of $\hat{\boldsymbol{\tau}}$ is neither excessively large nor small. Eq. (12) shows how small values of $\hat{\boldsymbol{\tau}}$ can cause gradients to explode. Eq. (13) shows how large values of $\hat{\boldsymbol{\tau}}$ from over-confident estimations can also lead to exploding gradients when the difference between $\boldsymbol{\Delta}$ and $\hat{\boldsymbol{\mu}}$ is large. In Eq. (11) we can also see how extremely small values of $\hat{\boldsymbol{\tau}}$ results in excessive errors being propagated over time, which in turn results in large gradients in both Eq. (12) and (13) (large $\boldsymbol{\Delta} - \hat{\boldsymbol{\mu}}$).

We use the following activation function in the output layer of $\hat{\tau}$ in order to constrain the value of $\hat{\boldsymbol{\tau}}$ to be in a given range $(\boldsymbol{\tau}_{min}, \boldsymbol{\tau}_{max})$:

$$f(\boldsymbol{h}) = (\boldsymbol{\tau}_{\max} - \boldsymbol{\tau}_{\min})\,\mathrm{Sig}(\boldsymbol{h}) + \boldsymbol{\tau}_{\min} \tag{14}$$

where $\mathrm{Sig}(\cdot)$ is the sigmoid function and $\boldsymbol{h}$ represents the pre-activations on the output layer. Because the output of the sigmoid function is in the range $(0, 1)$, the maximum value that $\hat{\boldsymbol{\tau}} = f(\boldsymbol{h})$ can take is $\boldsymbol{\tau}_{\max}$ when $\mathrm{Sig}(\boldsymbol{h}) = \mathbf{1}$. Likewise, the minimum value that $\hat{\boldsymbol{\tau}}$ can take is $\boldsymbol{\tau}_{\min}$ when $\mathrm{Sig}(\boldsymbol{h}) = \mathbf{0}$. Note that the sigmoid activation function $f(\boldsymbol{h})$ is only applied in the output layer. For the hidden layers, we used the ReLU activation function.

As a final line of defense for exploding gradients, we place gradient monitors between the unrolled networks. These monitors evaluate the norm of the gradients and reset the gradients to zero if the norm is larger than a predefined value. Fig. 3c shows the unrolling of the neural network with the gradient monitors for the multi-step training approach.

## *C. Long-term trajectory estimations using the learned DBNN model*

Trajectory optimization uses the long-term estimations of the DBNN model to find the control sequence that minimizes the control cost. In this case, the state trajectory is estimated using a standard sequential Monte-Carlo approach [22]. Given an initial state $\boldsymbol{z}_{[1]}$ and the list of control inputs $\left\{\boldsymbol{u}_{[t]}\right\}_1^{T_c}$, we estimate the system state trajectory using a set of $P$ particles, all starting from the initial $\boldsymbol{z}_{[1]}$ state. The state $\hat{\boldsymbol{z}}_{[t]}^{(p)}$ of each particle $p$ at time $t$ is estimated as follows:

$$\hat{\boldsymbol{z}}_{[t]}^{(p)} = \boldsymbol{z}_{[1]} + \sum_{k=1}^{t-1} \hat{\boldsymbol{\mu}}_{[k]}^{(p)} \odot \boldsymbol{\epsilon} \frac{1}{\sqrt{\hat{\boldsymbol{\tau}}_{[k]}^{(p)}}} \tag{15}$$

where:

$$\hat{\boldsymbol{\mu}}_{[k]}^{(p)} = \hat{\boldsymbol{\mu}}\left(\hat{\boldsymbol{z}}_{[k]}^{(p)} \oplus \boldsymbol{u}_{[k]}, w_\mu\right), \quad w_\mu \sim q_\phi(w)$$
$$\hat{\boldsymbol{\tau}}_{[k]}^{(p)} = \hat{\boldsymbol{\tau}}\left(\hat{\boldsymbol{z}}_{[k]}^{(p)} \oplus \boldsymbol{u}_{[k]}, w_\tau\right)$$

Eq. (15) is a standard unrolling of the neural-network. The expectations on the control cost in Eq. (8) are approximated using the sample mean of the particles:

$$\mathop{\mathbb{E}}_{\boldsymbol{z}_{[t]}}\left[\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right)\right] \approx \frac{1}{P}\sum_{p=1}^{P}\mathcal{L}_t\left(\hat{\boldsymbol{z}}_{[t]}^{(p)}, \boldsymbol{u}_{[t]}\right)$$

The standard deviation of the particles is used for measuring and visualizing the uncertainty of the predicted state trajectory.

## D. Computational complexity

The dominant operation in Algorithm 1 is the vector-matrix products performed during the DBNN forward and backpropagation computations. The complexity of these operations depends primarily on the size of the weight matrices. Assuming the networks $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\tau}})$ are composed by $(L_\mu, L_\tau)$ hidden layers, with $(M_\mu, M_\tau)$ hidden units in each layer, the number parameters of the DBNN can be roughly approximated as follows:

$$|w| = L_\mu\left(M_\mu^2\right) + L_\tau\left(M_\tau^2\right) \tag{16}$$

We use this expression to quantify the space complexity of the algorithm. The time complexity of forward and backward computations for a mini-batch of size $|\boldsymbol{X}|$ can be approximated as follows:

$$|\boldsymbol{X}|\, T_m\left(L_\mu\left(M_\mu^2\right) + L_\tau\left(M_\tau^2\right)\right) \tag{17}$$

where $T_m$ is the number of multi-step predictions. Long-term trajectory computations used for trajectory optimization (control) have an equivalent time complexity of $PT_c\left(L_\mu\left(M_\mu^2\right) + L_\tau\left(M_\tau^2\right)\right)$, where $P$ is the number of particles.

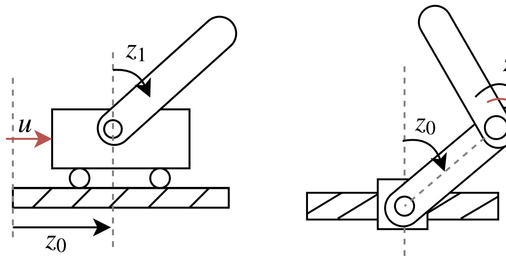
Eq. (16) and (17) demonstrate further potential benefits of using multi-step and heteroscedastic models. The dominant term in Eq. (16) and (17) is the square of hidden units $M_\mu^2$ and $M_\mu^2$. This shows that increasing the number of hidden units for a homoscedastic model is very expensive. On the other hand, increasing the number of time steps $T_m$ in multi-step training has a much lower cost regarding time complexity, and no cost in space complexity is incurred. For our application, space complexity is more valuable than time complexity because the optimizations are performed offline.

Eq. (16) also shows that adding a heteroscedastic model with $N$ hidden units is cheaper than adding such units into a homoscedastic model:

$$L_\mu\left(M_\mu^2\right) + L_\mu\left(N^2\right) < L_\mu\left(M_\mu + N\right)^2$$

## V. Experiments

Underactuated mechanical systems are commonly found in industrial applications [35] and provide interesting benchmark problems for controls. Therefore, for experimental evaluation, we selected the Cartpole (Fig. 4a) and Acrobot (Fig. 4b) as benchmark problems. In addition to having underactuated dynamics, these systems also have unstable fix-points and in the case of the Acrobot, chaotic behavior when no control is applied.



(a) Cartpole (b) Acrobot

Fig. 4. Illustration of the benchmark problems being considered

The *objectives* of the experiments are: 1) validate the use of the presented stochastic modeling framework for obtaining long-term predictions with uncertainty; 2) evaluate the advantages of using heteroscedastic models; 3) evaluate the advantages of multi-step training.

The *highlights* of the experimental evaluation are: 1) training using multi-step predictions provided better generalization and faster convergence rates in the trajectory optimization task, compared to single-step training; 2) heteroscedastic models provided estimations with lower uncertainty when compared to homoscedastic models; 3) the presented methodology is suitable for real-world applications; 4) multi-step training is stable, even when we use only one particle for the Monte-Carlo estimations during training. The experiments demonstrated that the methodology is able to provide accurate long-term predictions of non-linear underactuated systems. The methodology is suitable for real-world applications as the experiments demonstrate it is sample-efficient, i.e. it requires a relatively low number of samples from the real system. Furthermore, planning is performed completely offline. All expensive optimizations are executed offline, which means that they are not required to run in real-time.

## A. Experiment setup

In our experiments, the trajectory optimization task is to perform the swing-up maneuver for Cartpole and Acrobot. The maneuver entails driving the poles from a downward position at time $t = 0$, to an upward position at $t = T_c$.

The state for the Cartpole is four-dimensional $\boldsymbol{z} = [z_0, z_1, \dot{z_0}, \dot{z_1}]$, where $z_0 \in [-1, 1]$ represents the position of the cart and $z_1 \in [-4, 4]$ represents the position of the pole in radians. $\dot{z_0}$ and $\dot{z_1}$ represent the time derivatives of $z_0$ and $z_1$, respectively.

The state for the Acrobot is also four-dimensional $\boldsymbol{z} = [z_0, z_1, \dot{z_0}, \dot{z_1}]$, where $z_0 \in [-4, 4]$ represents the position of the first pole and $z_1 \in [-4, 4]$ the position of the second pole in radians.

For both Cartpole and Acrobot, the coordinate system was configured such that the $[0, 0, 0, 0]$ state corresponds to the poles in the upward position. The simulations were performed using OpenAI Gym [36] with PyBullet [37]

The objective of the trajectory optimization is to reach the target state $\boldsymbol{z}^*_{[T_c]} = [0, 0, 0, 0]$ at time $T_c$. To achieve this objective, the losses $\mathcal{L}_{T_c}, \mathcal{L}_t$ from Eq. (8) were defined as follows:

$$\mathcal{L}_{T_c}\left(\boldsymbol{z}_{[T_c]}, \boldsymbol{u}_{[T_c]}\right) = (\boldsymbol{z}_{[T_c]} - \boldsymbol{z}^*_{[T_c]})^T \boldsymbol{Q}_T (\boldsymbol{z}_{[T_c]} - \boldsymbol{z}^*_{[T_c]}) \quad (18)$$

$$\mathcal{L}_t\left(\boldsymbol{z}_{[t]}, \boldsymbol{u}_{[t]}\right) = \boldsymbol{u}^T_{[t]} \boldsymbol{Q}_u \boldsymbol{u}_{[t]} \quad (19)$$

where $\boldsymbol{Q}_T, \boldsymbol{Q}_u$ are diagonal matrices. $\mathcal{L}_{T_c}$ penalizes for reaching a state different than $\boldsymbol{z}^*_{[T_c]}$ at the end of the trajectory, while $\mathcal{L}_t$ penalizes for high control inputs.

We evaluated the performance of different DBNN architectures using two criteria: 1) The performance on the open-loop control task; 2) The quality of the DBNN long-term predictions.

Given that the control sequence $\boldsymbol{u}*$ obtained with Algorithm 1 is applied in open-loop, accurate long-term predictions are necessary to successfully reach the target state $\boldsymbol{z}^*_{[T]}$. We measured the quality of long-term trajectory predictions using: 1) the deviation of the real trajectory $\boldsymbol{z}$ from the predicted trajectory $\hat{\boldsymbol{z}}$; 2) the standard deviation of the estimations; 3) the containing-ratio of long-term predictions on a test dataset. The containing-ratio (CR-$n$) is the percentage of real samples inside $n-$ standard deviations.

We measured the deviation of the real trajectory ($\boldsymbol{z}_{[t]}$) from the predicted probability distribution ($\hat{\boldsymbol{z}}_{[t]}$) using the following metrics:

$$\delta_\mu(\boldsymbol{z}, \hat{\boldsymbol{z}}) = \frac{1}{T}\sum_{t=1}^{T} \|\boldsymbol{z}_{[t]} - \text{Mean}(\hat{\boldsymbol{z}}_{[t]})\|^2_2 \quad (20)$$

$$\delta_\sigma(\boldsymbol{z}, \hat{\boldsymbol{z}}) = \left\|\frac{1}{T}\sum_{t=1}^{T} \frac{|\boldsymbol{z}_{[t]} - \text{Mean}(\hat{\boldsymbol{z}}_{[t]})|}{\text{STD}(\hat{\boldsymbol{z}}_{[t]})}\right\| \quad (21)$$

where $\text{Mean}(\hat{\boldsymbol{z}}_{[t]})$ and $\text{STD}(\hat{\boldsymbol{z}}_{[t]})$ are the sample-mean and sample standard-deviation of the particles in Eq. (15). Eq. (21) provides a natural way to interpret the accuracy of the predictions in terms of a Standard Score (z-score). The goal is to obtain long-term estimations with low uncertainty. On the other hand, we do not want over-confident estimations, the real trajectory should be contained inside three standard deviations of the predicted distribution.

Table I lists the architectures considered in the experiments. OS stands for homoscedastic-single-step, ES heteroscedastic-single-step, OM homoscedastic-multi-step, and ES heteroscedastic-multi-step. The table shows the number of hidden units and layers used for $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\tau}}$. For example, $\hat{\boldsymbol{\mu}}[50, 50]$ means a DBNN with two hidden layers and 50 hidden units in each layer. For the Cartpole we used $p_{drop} = 0.1, T_c = 50$ and for the Acrobot $p_{drop} = 0.03, T_c = 110$. We used ReLU activation functions for the hidden layers.

To demonstrate the advantages of stochastic models over deterministic models, we compared the performance of the presented stochastic approach with a variation of the deterministic sequence-to-sequence LSTM model presented in [14]. Given that we consider only fully observable systems, instead of an LSTM encoder, we used a linear model to obtain the initial state of the LSTM decoder described in [14].

TABLE I
MODEL ARCHITECTURES

| Architecture | Cartpole $\mu$ | Cartpole $\tau$ | Cartpole $T_m$ | Acrobot $\mu$ | Acrobot $\tau$ | Acrobot $T_m$ |
|---|---|---|---|---|---|---|
| LSTM | $[10, 10]$ | - | 30 | $[25, 25]$ | - | 30 |
| OS (baseline) | $[50, 50]$ | - | - | $[100, 100]$ | - | - |
| ES | $[50, 50]$ | $[25]$ | - | $[100, 100]$ | $[50]$ | - |
| OM | $[50, 50]$ | - | 10 | $[100, 100]$ | - | 10 |
| EM | $[50, 50]$ | $[25]$ | 10 | $[100, 100]$ | $[50]$ | 10 |

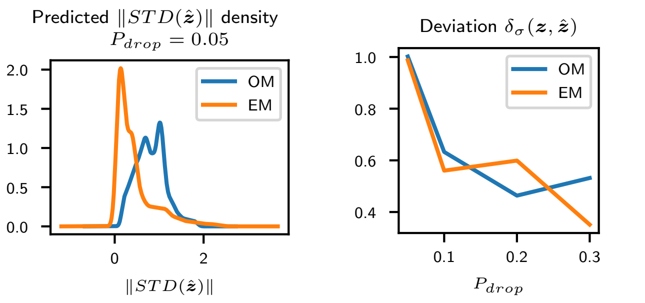
## B. Results

Table II shows the control performance achieved using the architectures from Table I. Given that the target state is located at $\boldsymbol{z}^* = \boldsymbol{0}$, we evaluate the control performance using: 1) the average absolute value of the final state $\left|\boldsymbol{z}^*_{T_c}\right|$; 2) the average magnitude of the final state $\|\boldsymbol{z}^*_{T_c}\|$. We also report the average control cost $\mathcal{L}_c$ of the real trajectory. Overall, Table II shows the EM model had the best performance on the control task for both Cartpole and Acrobot, with the lowest control cost $\mathcal{L}_c$. The deterministic LSTM had the worst performance in the control task for both systems.

Table III shows the quality of the estimated long-term trajectories. The table shows the STD magnitude and deviation of long-term estimations on: 1) a testing dataset, composed of trajectories that were not used to train the model; 2) the optimal trajectories found through iterations of Algorithm 1. Overall, EM model provided the estimations with the lowest uncertainty. EM model also provided the most accurate trajectories according to $\delta_\mu$.

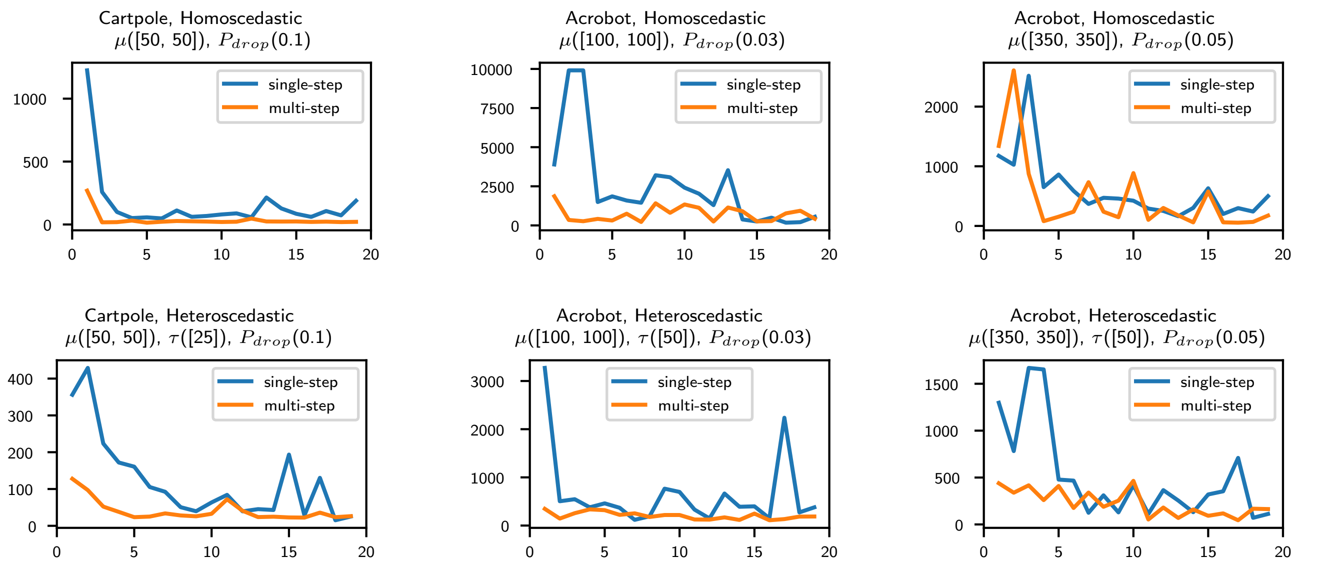
The LSTM model was unable to provide accurate long-term predictions. In addition to being unable to provide uncertainty estimations, LSTM was prone to over-fitting, which explains the relative small number of units used for this model and the higher number of $T_m$ used for training. Table III shows LSTM had the highest mean deviation $\sigma_\mu$ on the testing dataset.

Table IV shows the containing-ratios on the testing dataset, with $T = T_c$ time-step estimations. Table IV shows the



(a) Density distribution of $\|STD\|$ (b) Deviation $\delta_\sigma$ for different values of Dropout

Fig. 5. Comparison between OM and EM estimations of long-term trajectories (Cartpole) on testing dataset. a) OM provides estimations with lower uncertainty ($\|STD\|$ closer to zero) b) The deviation $\delta_\sigma$ is reduced by increasing dropout.



(a) Cartpole (b) Acrobot (small sized DBNNs) (c) Acrobot (large sized DBNNs)

Fig. 6. Final trajectory cost $\mathcal{L}_T$ for singe-step and multi-step training during iteration $j = [1, ..., 20]$ of Algorithm 1. Multi-step training improves the convergence rate. Optimal trajectories are found after few iterations, demonstrating the viability for real-world applications.

TABLE II
CONTROL PERFORMANCE OF ALGORITHM 1

| | Average $\lvert \boldsymbol{z}_{[T_c]} \rvert$ | | | | Average | Average |
|---|---|---|---|---|---|---|
| | $\lvert z_0 \rvert$ | $\lvert z_1 \rvert$ | $\lvert z_2 \rvert$ | $\lvert z_3 \rvert$ | $\lVert \boldsymbol{z}_{[T_c]} \rVert$ | $\mathcal{L}_c$ |
| **Cartpole** | | | | | | |
| LSTM | 0.37 | 1.37 | 1.05 | 5.29 | 5.73 | 25.785 |
| OS (baseline) | 0.52 | 0.85 | 2.66 | 4.56 | 5.99 | 15.995 |
| ES | 0.56 | 0.75 | 1.24 | 4.21 | 4.98 | 15.061 |
| OM | 0.25 | 0.35 | 2.05 | 1.55 | 2.96 | 5.898 |
| EM | 0.41 | 0.26 | 2.75 | 1.04 | 3.07 | **5.298** |
| **Acrobot** | | | | | | |
| LSTM | 2.65 | 2.51 | 3.92 | 6.48 | 10.22 | 197.366 |
| OS (baseline) | 1.36 | 3.10 | 2.68 | 1.80 | 5.34 | 81.991 |
| ES | 1.42 | 1.76 | 2.84 | 2.68 | 5.02 | 67.654 |
| OM | 0.99 | 1.18 | 3.34 | 4.65 | 6.50 | 60.519 |
| EM | 0.50 | 1.61 | 2.45 | 2.74 | 4.56 | **21.025** |

TABLE III
LONG-TERM PREDICTION PERFORMANCE ON TEST DATASET AND OPTIMAL TRAJECTORIES

| | Test dataset | | | Optimal trajectory $\boldsymbol{z}^*$ | | |
|---|---|---|---|---|---|---|
| | $\delta_\mu$ | $\lVert STD \rVert$ | $\delta_\sigma$ | $\delta_\mu$ | $\lVert STD \rVert$ | $\delta_\sigma$ |
| **Cartpole** | | | | | | |
| LSTM | 0.367 | - | - | 5.59 | - | - |
| OS (baseline) | 0.252 | 2.96 | 0.72 | 5.59 | 0.60 | 1.03 |
| ES | 0.282 | 0.77 | 0.74 | 3.15 | 0.23 | 1.20 |
| OM | 0.086 | 1.46 | 0.56 | 4.84 | 0.52 | **0.99** |
| EM | **0.071** | **0.43** | **0.42** | **1.55** | **0.21** | 1.01 |
| **Acrobot** | | | | | | |
| LSTM | 12.71 | - | - | 48.74 | - | - |
| OS (baseline) | 3.70 | 27.91 | 0.85 | 65.53 | 3.26 | 0.67 |
| ES | 1.76 | 6.64 | 0.87 | 17.75 | 0.85 | 1.45 |
| OM | 1.47 | 13.88 | **0.57** | 34.69 | 2.28 | **0.60** |
| EM | **0.80** | **4.15** | 0.59 | **6.19** | **0.68** | 1.15 |

containing-ratios for $1\sigma$ (CR-1), $2\sigma$ (CR-2), and $3\sigma$ (CR-3), where $\sigma$ stands for standard deviation. The table also shows the skewness of the test samples after computing their corresponding z-score. This table allows us to compare the quality of the uncertainty estimations provided by the different models. Overall, we see that the containing-ratios approximate the (68-95-99.7) rule of the Normal distribution, with the EM model providing the best CRs.

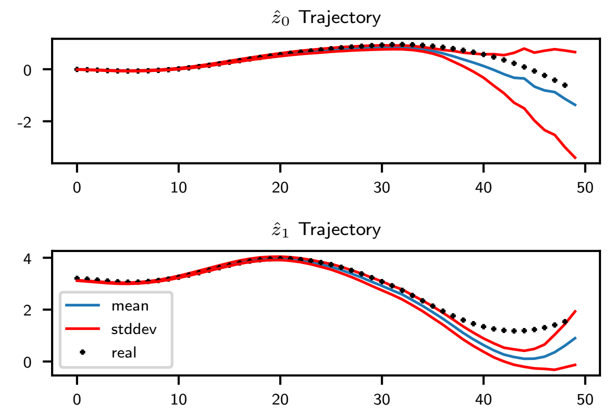
## *C. Comparative analysis: Heteroscedastic vs Homoscedastic*

Heteroscedastic models provided estimations with lower uncertainty compared to homoscedastic models. Table III shows the estimations of ES and EM models have lower standard deviations compared to OS and OM models. For the Acrobot model, the reduction is considerable, a result that can be attributed to the higher difficulty posed by the Acrobot dynamics. Fig. 5a shows the distribution of estimated $\lVert STD \rVert$ values on the test dataset. This figure shows in more detail the lower uncertainty provided by heteroscedastic models.

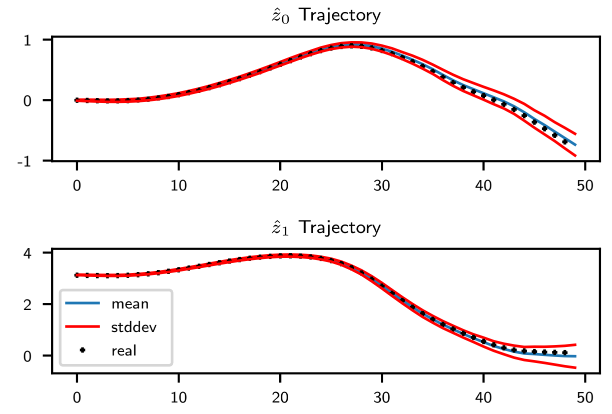
TABLE IV
CONTAINING-RATIOS OF LONG-TERM PREDICTIONS ON TEST DATASET

| | CR-1 | CR-2 | CR-3 | skewness |
|---|---|---|---|---|
| **Cartpole** | | | | |
| OS (baseline) | 0.52 | 0.96 | 1.00 | [-0.09 , 0.05 , -0.71 , 0.34] |
| ES | 0.78 | 0.94 | 0.96 | [-4.92 , -1.68 , -10.87, -1.26] |
| OM | 0.67 | 0.97 | 1.00 | [ 0.09 , -0.61 , -0.07, -0.33] |
| EM | 0.79 | 1.00 | 1.00 | [ 0.21 , 0.92 , 1.27, 2.35] |
| **Acrobot** | | | | |
| OS (baseline) | 0.35 | 0.91 | 0.99 | [ 0.33 , 0.13 , 0.05 , 0.13 ] |
| ES | 0.47 | 0.93 | 0.99 | [ -0.09 , 0.12 , -0.02 , -0.29 ] |
| OM | 0.58 | 0.97 | 1.00 | [ 0.81 , 0.29 , 0.29 , 0.23 ] |
| EM | 0.70 | 0.95 | 0.99 | [ 0.10 , 0.55 , 0.30 , -0.10 ] |

Although heteroscedastic models provided estimations with lower uncertainty, the experiments also showed that het-

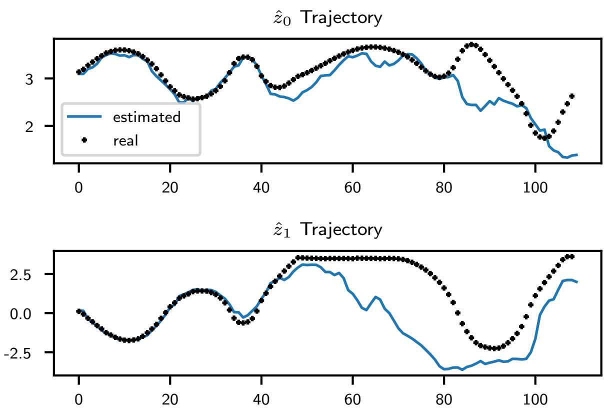


(a) Estimations during first iterations have high uncertainty ($\|STD\|$)

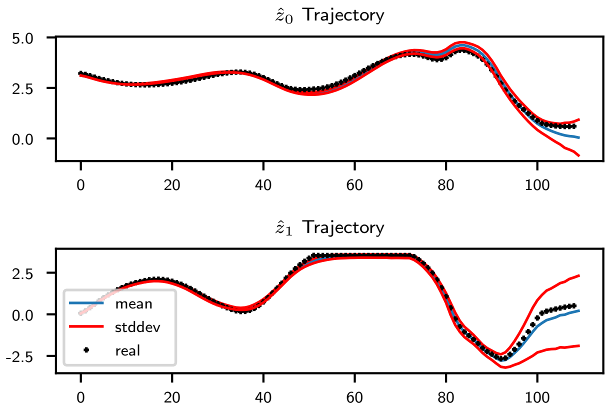


(b) Estimations on final iterations have low uncertainty ($\|STD\|$)

Fig. 7. Estimated optimal trajectories found during the execution of Algorithm 1 for the Cartpole. The standard deviation is used to quantify and visualize the uncertainty. The algorithm is able to find optimal open-loop trajectories, providing accurate long-term predictions of the system trajectory with low uncertainty.



(a) Model-mismatch when using deterministic LSTM model. The estimated trajectory deviates from the real trajectory from t=45



(b) EM provides accurate long-term estimations that match the trajectory of the real system

Fig. 8. Long-term optimal trajectory estimations provided by EM and LSTM models for the Acrobot. Deterministic LSTM is unable to provide accurate long-term predictions.

eroscedastic models are more likely to provide overconfident estimations. Table IV shows that for the Cartpole, ES had the lowest CR3 score and the worst skewness.

Fig. 5b shows how increasing Dropout can be used to reduce the deviation $\delta_\sigma(\boldsymbol{z}, \hat{\boldsymbol{z}})$, reducing the chance of obtaining overconfident estimations. This serves as a tool for alleviating potential model-mismatch problems.

## D. *Comparative analysis: single-step vs multi-step training*

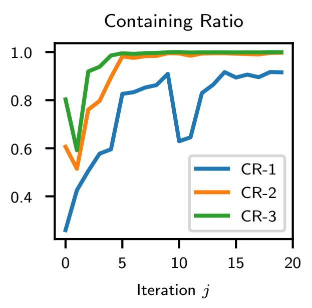
Multi-step training had the advantage of improving generalization, providing accurate long-term predictions with lower deviation and better containing-ratios in the testing dataset. The combination of heteroscedastic and multi-step training provided the best performance in the control (Table II) and estimation tasks (Tables III and IV).

Tables III and IV serve as evidence of the improved generalization of multi-step training. Table III shows that multi-step training reduces the deviation ($\delta_\mu$ and $\delta_\sigma$) between the predictions and the real behavior of the system. EM models provided accurate estimations with the lowest uncertainty ($\|STD\|$) without being overconfident (low deviation and best containing-ratios).

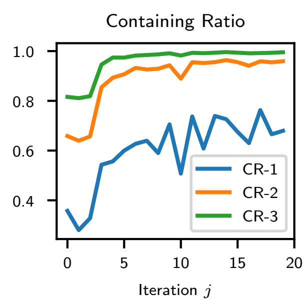
Table IV shows that multi-step training improves the containing-ratios of long-term predictions. This is of particular interest when using heteroscedastic models. Multi-step training alleviates the problem of overconfident estimations given by ES models. The best containing-ratios correspond to EM models.

Fig. 6 shows a comparison of the performance between single-step and multi-step training using different architectures. The figure shows the value for the final control cost $\mathcal{L}_{[T_c]}$ on each iteration $j$ of Algorithm 1. Fig. 6 shows that for both, Cartpole and Acrobot, multi-step training provides faster convergence rates for the trajectory optimization task.

Fig. 6 also shows the viability of the algorithm for real-world control applications. By iteration $j = 7$ most of the experiments had converged to an optimal trajectory. For the Cartpole (Fig. 6a), the algorithm converges after the first couple iterations. For the experiments, we used $U_o = 20$ initial trajectories and $U = 15$ trajectories were collected from the simulation in each iteration of Algorithm 1. Overall,



(a) Cartpole



(b) Acrobot

Fig. 9. Containing-ratio of long-term test trajectories during iteration $j$ of Algorithm 1. The containing-ratios approximate the (68-95-99.7) rule after few iterations.

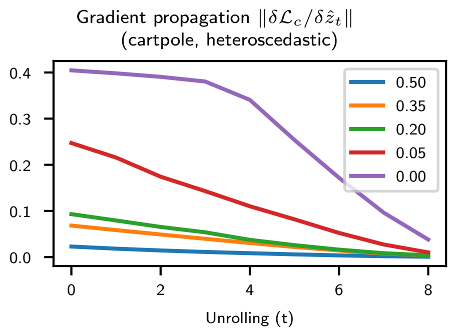


Fig. 10. Gradient propagation for different values of Dropout. We observe a stable increase of the gradient magnitude as the gradients are back-propagated

TABLE V
BEST VALUES OF FINAL STATE $\boldsymbol{z}_{[T]}$ AFTER EXECUTING THE OPTIMAL TRAJECTORY USING EM MODEL.

| System | $z_0$ | $z_1$[rad] | $z_2$ | $z_3$[rad/s] |
|---|---|---|---|---|
| Cartpole | -0.40 [m] | 0.03 | -2.27 [m/s] | -0.47 |
| Acrobot | -0.22 [rad] | -0.39 | -1.39 [rad/s] | -0.46 |

the algorithm only required 125 trajectories sampled from the system to find an optimal solution.

When compared to homoscedastic models, Fig. 6b and 6c show that heteroscedastic models allowed us to use smaller DBNNs. Fig. 6c shows that the performance of OM models can be improved by increasing the number of hidden layers. However, this comes at the expense of higher computational complexity. Using small DBNNs has the advantage of reduced memory requirements for the optimization tasks.

## E. Long-term predictions using EM models

Fig. 7 shows the predictions of the state trajectory for the Cartpole during first (7a) and final (7b) iterations of Algorithm 1. We used the standard deviation of the predictions to visualize the uncertainty. Fig. 7a shows that the algorithm starts with high uncertain predictions. After convergence, Fig. 7b shows the trajectory estimations are made with low uncertainty.

Fig. 8 shows a comparison of the long-term trajectory estimations provided by LSTM and EM models. Fig. 8a shows that LSTM models suffer from model mismatch problems, where the estimated trajectory deviates from the real trajectory. In contrast, Fig. 8b shows the EM model is able to provide accurate long-term predictions.

Table V presents the best values of the final state $\boldsymbol{z}_{[T]}$ during the execution of Algorithm 1. Table V and Fig. 8b show that Algorithm 1 is able to find open-loop trajectories that drive the system close to the target state ($\boldsymbol{z} = 0$).

Fig. 9 shows the containing-ratios on the test dataset in each iteration $j$ of Algorithm 1. Although the quality of the uncertainty estimation is poor in the first iteration, the figure shows the containing-ratios quickly approximate the (68-95-99.7) rule of the Normal distribution.

## F. Gradient propagation

Unstable gradient propagation poses a challenge for multi-step training. Appropriate selection of $(\boldsymbol{\tau}_{min},\ \boldsymbol{\tau}_{max})$ and weight initialization allowed us to achieve stable optimization. The bounds $(\boldsymbol{\tau}_{min}, \boldsymbol{\tau}_{max})$ were necessary to stabilize multi-step training and considerably improved the performance of the models trained using single-step approach.

Fig. 10 shows the magnitude of the backpropagated gradients for different dropout probabilities. The gradients grow steadily as they are back-propagated. Increasing the dropout probability reduced the rate in which gradients grow. The figure shows how gradients grow quickly without dropout, reaching the point where the gradient monitors are activated, preventing excessively large gradients from destabilizing the training.

## G. Discussion

In this paper we covered the first two stages for controlling underactuated systems: modeling and open-loop planning. The focus of the presented approach was to study the performance of DBNN models on providing accurate long-term estimations for open-loop planning. The accuracy of the learned stochastic model allowed us to plan completely offline and execute the trajectory in open-loop. This approach allowed us to move all computationally expensive optimizations offline and make the presented approach applicable on real-world scenarios.

However, applying the trajectory in open-loop has two challenges: 1) performance degrades for excessively long trajectories; 2) control is sensitive to external disturbances. As shown in Fig. 7b and 8b, as time increases, the uncertainty is also increasing. This behavior is a design choice that results from propagating the uncertainty overtime in Eq. (6). Due to process noise and approximation errors, the uncertainty should increase over time. However, this results in estimations with low confidence when planning over excessively long trajectories.

In order to further improve control performance and guarantee robustness against external noise, trajectory stabilization techniques can be used on top of our approach. Techniques such as Time-Varying LQR feedback stabilization [1] can be introduced to stabilize the system around the optimal open-loop trajectory found with Algorithm 1.

## VI. Conclusion

In this paper, we presented an approach for modeling and planning under uncertainty using Deep Bayesian Neural-Networks. We presented a method for learning dynamics using multi-step predictions. The approach includes different tools for ensuring stable learning of the dynamics for heteroscedastic models. The learned model was successfully used in a trajectory optimization task.

The presented data-driven modeling and planning approach was able to find optimal trajectories that can perform the swing-up maneuver for both Cartpole and Acrobot, without the need for expert knowledge of the dynamics. The learned stochastic model was able to accurately estimate long-term state trajectories together with the uncertainty of the predictions. Compared to single-step training, multi-step training showed improved generalization and faster convergence rates in the trajectory optimization task. The accuracy of the estimated trajectory probability distributions allowed us to plan completely offline and execute the optimal trajectory in open-loop.

The success of the presented multi-step training approach, which uses single Monte-Carlo particles, is particularly interesting for future research, especially for partially observable systems. The use of single particles provides an efficient approach for training the model using back-propagation and mini-batch stochastic optimization. Future research will be conducted on extending and evaluating the presented model for partially-observable systems. Furthermore, trajectory stabilization techniques will be investigated in order to improve control performance and ensure robustness against external disturbances.

## Appendix A
## Variational Distributions

Assuming $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\tau}}$ are composed by $L_\mu$ and $L_\tau$ hidden layers, respectively, the variational distributions for the parameters are defined as follows:

$$w_\mu = \left\{\boldsymbol{A}^{(l)}, \boldsymbol{a}^{(l)}\right\}_{l=1}^{L_\mu}; \quad w_\tau = \left\{\boldsymbol{B}^{(l)}, \boldsymbol{b}^{(l)}\right\}_{l=1}^{L_\tau}$$
$$q_\phi\left(\boldsymbol{A}^{(l)}_{[:,i]}\right) = P_{drop}\mathcal{N}\left(0, \sigma_a^2\boldsymbol{I}\right) + (1-P_{drop})\mathcal{N}\left(\boldsymbol{A}^{(l)}_{\phi[:,i]}, \sigma_A^2\boldsymbol{I}\right)$$
$$q_\phi\left(\boldsymbol{a}^{(l)}\right) = \mathcal{N}\left(\boldsymbol{a}^{(l)}_\phi, \sigma_a^2\boldsymbol{I}\right)$$
$$q_\phi\left(\boldsymbol{B}^{(l)}_{[:,i]}\right) = \mathcal{N}\left(\boldsymbol{B}^{(l)}_{\phi[:,i]}, \sigma_B^2\boldsymbol{I}\right); \; q_\phi\left(\boldsymbol{b}^{(l)}\right) = \mathcal{N}\left(\boldsymbol{b}^{(l)}_\phi, \sigma_b^2\boldsymbol{I}\right)$$

where $\boldsymbol{A}, \boldsymbol{a}$ represent the weights and bias of $\hat{\boldsymbol{\mu}}$ and $\boldsymbol{B}, \boldsymbol{b}$ the weights and bias of $\hat{\boldsymbol{\tau}}$. The notation $\boldsymbol{A}^{(l)}_{[:,i]}$ represents the $i$th column of $\boldsymbol{A}^{(l)}$. We follow [28] to define the distribution of $q_\phi\left(\boldsymbol{A}^{(l)}_{[:,i]}\right)$ as a mixture of Gaussians with dropout probability $P_{drop}$.

The priors for the parameters were defined as follows:

$$p\left(\boldsymbol{A}^{(l)}_{[:,i]}\right) = \mathcal{N}\left(0, \beta_A^2\boldsymbol{I}\right); \; p\left(\boldsymbol{B}^{(l)}_{[:,i]}\right) = \mathcal{N}\left(0, \beta_B^2\boldsymbol{I}\right)$$

We do not regularize the bias parameters, hence no priors are placed on $\boldsymbol{a}^{(l)}$ or $\boldsymbol{b}^{(l)}$.

To simplify the model, we choose not to optimize over the variance parameters of the variational distributions. Given that the variational distribution and priors are defined using Gaussian distributions, the KL divergence is computed as follows:

$$KL(q_\phi(w)|p(w)) = \sum_l \frac{\left\|\boldsymbol{A}^{(l)}\right\|_F^2}{\beta_A^2} + \sum_l \frac{\left\|\boldsymbol{B}^{(l)}\right\|_F^2}{\beta_B^2} + K$$

where $K$ is a constant that is ignored during optimization and $\|\boldsymbol{A}\|_F$ is the Frobenius norm. This expression can be derived directly from the KL divergence between Gaussians and the approximation presented in [28] for dropout.

During inference, we use the mean of the distributions directly as single point estimates. This allows us to represent the following weights and biases directly as variational parameters: $\boldsymbol{a}^{(l)} = \boldsymbol{a}^{(l)}_\phi$, $\boldsymbol{B}^{(l)} = \boldsymbol{B}^{(l)}_\phi$, $\boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)}_\phi$.

Furthermore, the weight matrices of $\hat{\boldsymbol{\mu}}$ can be reparameterized using a Bernoulli distribution:

$$\boldsymbol{A}^{(l)} \sim \boldsymbol{A}^{(l)}_\phi \text{diag}\left(\boldsymbol{\alpha}\right)$$

where $\alpha_i \sim \text{Bern}\left(P_{drop}\right)$ [28]. Finally, the trainable variational parameters $\phi = \phi_\mu \cup \phi_\tau$ for heteroscedastic models are:

$$\phi_\mu = \left\{\boldsymbol{A}^{(l)}_\phi, \boldsymbol{a}^{(l)}_\phi\right\}_{l=1}^{L_\mu}; \; \phi_\tau = \left\{\boldsymbol{B}^{(l)}_\phi, \boldsymbol{b}^{(l)}_\phi\right\}_{l=1}^{L_\tau}$$

For homoscedastic models, $\phi_\tau = \{\boldsymbol{\tau}_m\}$.

## Appendix B
## Loss Gradients

The gradients of $\mathcal{L}_R$ presented in Eq. (12) and (13) can be obtained using the following differentiation rules [38]:

$$\frac{\partial\left(\ln|\boldsymbol{\Omega}|\right)}{\partial\boldsymbol{\Omega}} = \boldsymbol{\Omega}^{-T}; \quad \frac{\partial\left(\boldsymbol{v}^T\boldsymbol{\Omega}\boldsymbol{v}\right)}{\partial\boldsymbol{\Omega}} = \boldsymbol{v}\boldsymbol{v}^T \tag{22}$$
$$\frac{\partial\left(\boldsymbol{v}^T\boldsymbol{\Omega}\boldsymbol{v}\right)}{\partial\boldsymbol{v}} = \left(\boldsymbol{\Omega} + \boldsymbol{\Omega}^T\right)\boldsymbol{v}$$

Let $\boldsymbol{v} = (\boldsymbol{y} - \hat{\boldsymbol{\mu}})$ and $\boldsymbol{\Omega} = \text{diag}\left(\hat{\boldsymbol{\tau}}\right)$, i.e. $\boldsymbol{\Omega}$ is a diagonal matrix where the diagonal is equal to $\hat{\boldsymbol{\tau}}$. We can express the loss $\mathcal{L}_R$ from Eq. (5) as follows:

$$\mathcal{L}_R = -\log|\boldsymbol{\Omega}| + \boldsymbol{v}^T\boldsymbol{\Omega}\boldsymbol{v}$$

where $\boldsymbol{\Omega}$ is symmetric ($\boldsymbol{\Omega} = \boldsymbol{\Omega}^T$). Using the differentiation rules from Eq. (22), the gradients of $\mathcal{L}_R$ can be expressed as follows:

$$\frac{\partial\mathcal{L}_R}{\partial\boldsymbol{\Omega}} = -\boldsymbol{\Omega}^{-1} + \boldsymbol{v}\boldsymbol{v}^T; \quad \frac{\partial\mathcal{L}_R}{\partial\boldsymbol{v}} = 2\boldsymbol{\Omega}\boldsymbol{v}$$

The gradient $\frac{\partial\mathcal{L}_R}{\partial\hat{\boldsymbol{\tau}}}$ in Eq. (12) is equal to the diagonal of $\frac{\partial\mathcal{L}_R}{\partial\boldsymbol{\Omega}}$. Finally, by using the chain rule we obtain the gradient $\frac{\partial\mathcal{L}_R}{\partial\hat{\boldsymbol{\mu}}} = -\frac{\partial\mathcal{L}_R}{\partial\boldsymbol{v}}$ which is equal to the gradient shown in Eq. (13).

## References

[1] R. Tedrake, "LQR-trees: Feedback motion planning on sparse randomized trees," in *Proc. of Robotics: Science and Systems V*, Seattle, USA, June 2009.

[2] ——. (2018) Underactuated robotics: Algorithms for walking, running, swimming, flying, and manipulation. [Online]. Available: http://underactuated.mit.edu/

[3] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 2, pp. 408–423, Feb 2015.

[4] W. Zine, Z. Makni, E. Monmasson, L. Idkhajine, and B. Condamin, "Interests and limits of machine learning-based neural networks for rotor position estimation in ev traction drives," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 1942–1951, May 2018.

[5] Z. Wang, C. Hu, Y. Zhu, S. He, K. Yang, and M. Zhang, "Neural network learning adaptive robust control of an industrial linear motor-driven stage with disturbance rejection ability," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2172–2183, Oct 2017.

[6] C. Yang, Y. Jiang, Z. Li, W. He, and C. Su, "Neural control of bimanual robots with guaranteed global stability and motion precision," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1162–1171, June 2017.

[7] N. K. Dhar, N. K. Verma, and L. Behera, "Adaptive critic-based event-triggered control for hvac system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 1, pp. 178–188, Jan 2018.

[8] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7559–7566.

[9] D. Marino, K. Amarasinghe, M. Anderson, N. Yancey, Q. Nguyen, K. Kenney, and M. Manic, "Data driven decision support for reliable biomass feedstock preprocessing," in *2017 Resilience Week (RWS)*, Sep. 2017, pp. 97–102.

[10] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[12] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. A. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," *arXiv:1707.02286 [cs.AI]*, July 2017.

[13] M. Khodayar, O. Kaynak, and M. E. Khodayar, "Rough deep neural architecture for short-term wind speed forecasting," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 2770–2779, Dec 2017.

[14] D. L. Marino, K. Amarasinghe, and M. Manic, "Building energy load forecasting using deep neural networks," in *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Oct 2016, pp. 7046–7051.

[15] L. Wen, X. Li, L. Gao, and Y. Zhang, "A new convolutional neural network-based data-driven fault diagnosis method," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, pp. 5990–5998, July 2018.

[16] H. Hu, B. Tang, X. Gong, W. Wei, and H. Wang, "Intelligent fault diagnosis of the high-speed train with big data based on deep neural networks," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2106–2116, Aug 2017.

[17] W. Lu, B. Liang, Y. Cheng, D. Meng, J. Yang, and T. Zhang, "Deep model based domain adaptation for fault diagnosis," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 3, pp. 2296–2305, March 2017.

[18] 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," in *4th International Conference on Learning Representations (ICLR)*, May 2016.

[19] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[20] F. Guo, H. Kodamana, Y. Zhao, B. Huang, and Y. Ding, "Robust identification of nonlinear errors-in-variables systems with parameter uncertainties using variational bayesian approach," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3047–3057, Dec 2017.

[21] R. Frigola, Y. Chen, and C. E. Rasmussen, "Variational gaussian process state-space models," in *Proc. of the 27th International Conference on Neural Information Processing Systems*, 2014, pp. 3680–3688.

[22] Y. Gal, R. T. McAllister, and C. E. Rasmussen, "Improving pilco with bayesian neural network dynamics models," in *ICML Workshop on Data-Efficient Machine Learning*, June 2016.

[23] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv:1702.01182 [cs.LG]*, Feb 2017.

[24] M. Al-Shedivat, A. G. Wilson, Y. Saatchi, Z. Hu, and E. P. Xing, "Learning scalable deep kernels with recurrent structure," *Journal of Machine Learning Research*, vol. 18, no. 82, pp. 1–37, 2017.

[25] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, "Learning and policy search in stochastic dynamical systems with bayesian neural networks," in *5th International Conference on Learning Representations (ICLR)*, April 2017.

[26] T. M. Moerland, J. Broekens, and C. M. Jonker, "Learning multi-modal transition dynamics for model-based reinforcement learning," *arXiv:1705.00470 [stat.ML]*, Aug 2017.

[27] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[28] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proc. of the 33nd International Conference on Machine Learning (ICML)*, June 2016, pp. 1050–1059.

[29] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations (ICLR)*, 2014.

[30] D. Hein, S. Depeweg, M. Tokic, S. Udluft, A. Hentschel, T. A. Runkler, and V. Sterzing, "A benchmark environment motivated by industrial control problems," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Nov 2017, pp. 1–8.

[31] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, May 2015.

[32] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct 1990.

[33] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks." in *Proc. of the 30th International Conference on Machine Learning (ICML)*, vol. 28, 2013, pp. 1310–1318.

[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, May 2010, pp. 249–256.

[35] N. Sun, T. Yang, Y. Fang, B. Lu, and Y. Qian, "Nonlinear motion control of underactuated three-dimensional boom cranes with hardware experiments," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 3, pp. 887–897, March 2018.

[36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv:1606.01540 [cs.LG]*, June 2016.

[37] E. Coumans and Y. Bai. (2016) Pybullet, a python module for physics simulation for games, robotics and machine learning. [Online]. Available: https://pybullet.org/

[38] K. B. Petersen and M. S. Pedersen. (2008) The matrix cookbook. [Online]. Available: https://www.math.uwaterloo.ca/%7Ehwolkowi/matrixcookbook.pdf

**Daniel L. Marino** (marinodl@vcu.edu) received his B.Eng. in automation engineering from La Salle University, Colombia, in 2015. He is currently a research assistant and a doctoral student at Virginia Commonwealth University. His research interests include stochastic modeling, deep learning and optimal control.

**Milos Manic** (misko@ieee.org) Milos Manic (misko@ieee.org) (SM'06-M'04-StM'96) received the Dipl.Ing. and M.S. degrees in electrical engineering and computer science from the University of Niš, Niš, Serbia in 1991 and 1997 respectively, and the Ph.D. degree in computer science from the University of Idaho in 2003. Dr. Manic is a Professor with Computer Science Department and Director of VCU Cybersecurity Center at Virginia Commonwealth University. He completed over 30 research efforts in the area of data mining and machine learning applied to cybersecurity, critical infrastructure protection, energy security, and resilient intelligent control. Dr. Manic has given over 30 invited talks around the world, authored over 180 refereed articles in international journals, books, and conferences, holds several U.S. patents and has won 2018 R&D 100 Award for Autonomic Intelligent Cyber Sensor (AICS). He is an officer of IEEE Industrial Electronics Society, founding chair of IEEE IES Technical Committee on Resilience and Security in Industry, and general chair of IEEE IECON 2018, IEEE HSI 2019.