STRUCTURED MECHANICAL MODELS FOR EFFICIENT REINFORCEMENT LEARNING

Kunal Menda*, Jayesh K. Gupta*, Zachary Manchester & Mykel J. Kochenderfer
Stanford University
{kmenda,jayeshkg,zacmanchester,mykel}@stanford.edu

ABSTRACT

Learning accurate dynamics models is necessary for optimal, compliant control of robotic systems. Model-based reinforcement learning attempts to learn accurate models by interacting with the environment, and the efficiency of doing so depends on how the model is parameterized. Current approaches to white-box modeling using analytic parameterizations, or black-box modeling using neural networks, can suffer from high bias or high variance. We address the need for a flexible, gray-box model of mechanical systems that can seamlessly incorporate prior knowledge where it is available, and train expressive function approximators where it is not. We propose to parameterize a mechanical system using neural networks to model its Lagrangian and the generalized forces that act on it. We test our method on a simulated, actuated double pendulum. We show that our method outperforms a black-box model in terms of data-efficiency, as well as performance in model-based reinforcement learning.

1 INTRODUCTION

Model-based reinforcement learning attempts to solve a new task by first acquiring an accurately model of environment dynamics, and then synthesizing a controller based on this model to solve the task. When specifying the model, one faces the perennial question of whether to seek out domain expertise, or to take a data-driven, black-box approach to constructing such a model. The former approach would make assumptions about the system, such as its kinematic structure, inertia properties, and assumptions regarding the forces acting on the system, leaving only a few parameters for data-driven calibration (An et al., 1985; Åström & Eykhoff, 1971). The latter approach (Werbos et al., 1992; Raissi et al., 2018; Chen et al., 1990), on the other hand, would treat the system’s equations of motion as any other function that the tools of machine learning are capable of fitting. That is, this approach would reduce the problem of learning the system dynamics to that of optimizing the parameters of an expressive function class, such as a neural network, in order to minimize some form of a prediction loss.

Both of the aforementioned approaches have limitations, summarized in Figure 1. The assumptions made by the domain expert may not capture hard-to-model effects, leading to inaccuracies via *model bias*. On the other hand, while the black-box approach of training an overparameterized function class may be capable of capturing the phenomena present in the training data, it often requires infeasibly large amounts of training data to achieve generalization, due to *model variance*.

Ideally, we would want to take a *gray-box* approach that models the parts of the system for which we believe that models are accurate, while capturing the difficult-to-model system dynamics by training a highly expressive function class such as a neural network from data. However, typical approaches to fitting system dynamics with neural networks do not allow us to easily incorporate prior knowledge. Although there have been proposals to learn *offset functions* that correct for model bias, they are still restricted to specific use cases Ratliff et al. (2016); Nguyen-Tuong & Peters (2010).

In this work, we propose a structured approach to gray-box modeling of mechanical systems. Instead of treating the equations of motion governing the system as an arbitrary functional mapping, we make the single assumption that the system conforms to *Lagrangian dynamics*. Consequently, we propose to learn the *Lagrangian* of the system, as well as a structured representation of the forces that act on the system. By doing so, we can parameterize the space of all mechanical systems in a structured and modular manner. From these two functions, we can evaluate the accelerations acting on the system by using the *Euler-Lagrange equation*. Ahmadi et al. (2018) and Lutter et al. (2019) also propose formulations for learning the Lagrangian of a system, but assume all forces acting on the system to be directly measurable as opposed to learned functions of the state and control input.

The method we present has the flexibility to incorporate as much prior knowledge as in a white-box approach, and, in the event of having no prior knowledge whatsoever, the method remains as expressive as a black-box approach.

*Authors contributed equally.

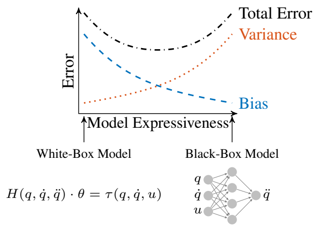

Figure 1: The bias-variance tradeoff as a function of model expressiveness.

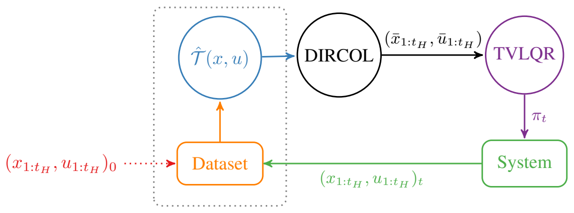
$\hat{\mathcal{T}}(x,u)$
DIRCOL
$(\bar{x}_{1:t_H},\bar{u}_{1:t_H})$
TVLQR
π_t
$(x_{1:t_H},u_{1:t_H})_0$
Dataset
System
$(x_{1:t_H},u_{1:t_H})_t$

Figure 2: Framework for Model-Based Reinforcement Learning

However, we will show that even in this scenario, our method achieves lower model variance than naive approaches to black-box modeling owing to the constraints of physical compliance.

By modularizing the hypothesis class into functions that represent the system's Lagrangian and functions that represent the forces acting on it, we gain the principal benefit of being able to incorporate prior knowledge where it is available. For example, say we only know *a priori* that the system is *control affine* (i.e., there exists a linear mapping between actuator inputs and the torques applied to the system) and that no forces other than gravity act on the system. In such a situation, we can use an expressive function class to model the system's Lagrangian, while using a much more restricted function class to model the control-affine torques, thereby reducing model variance.

2 Background

Our work draws inspiration from a variety of fields including classical mechanics, system identification, model-based control, and modern machine learning techniques such as deep learning.

2.1 Lagrangian Dynamics and Prediction

The purpose of modeling a dynamic system is to be able to predict how the state of the system evolves over time. Formally, the state of a system is described using *generalized coordinates* $q \in \mathbb{R}^N$ and velocities $\dot{q} \in \mathbb{R}^N$, where N is the number of coordinates.

Defining Lagrangian as a difference of kinetic energy T and potential energy V, the dynamics of the system are specified by the Euler-Lagrange (EL) equation:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} = F(q, \dot{q}, u) \tag{1}$$

where $F(q, \dot{q}, u)$ represents the *generalized forces* that act on the system, and $u \in \mathbb{R}^M$ are the actuator inputs. For a mechanical system, applying the chain-rule gets us the commonly referred *manipulator equation* (Murray et al., 1994):

$$\mathbf{M}(q)\ddot{q} + \mathbf{C}(q, \dot{q})\dot{q} + G(q) = F(q, \dot{q}, u) \tag{2}$$

Here, $G(q) = -\nabla_q V(q)$ are the conservative forces that act on the system, and $\mathbf{C}(q, \dot{q})$ is the *Coriolis matrix* of the system (Murray et al., 1994). $\mathbf{C}(q, \dot{q})$ can be derived from the *mass-matrix* $\mathbf{M}(q)$, given q and $\dot{q}$. Hence, if we have a function predicting the mass matrix $\mathbf{M}(q)$, the potential $V(q)$, and the forces $F(q, \dot{q}, u)$ that act on the system, we have fully specified the continuous time dynamics of the system.

Thus, given $\mathbf{M}(q)$, $V(q)$, and $F(q, \dot{q}, u)$, as well as initial conditions $q_t, \dot{q}_t$ and actuator input u_t, we can simulate the model forward in time by integrating the Equation (2) to predict $q_{t'}, \dot{q}_{t'}$ for some $t' = t + \Delta t$.

2.2 System Identification

The inverse problem of finding $\mathbf{M}(q)$, $V(q)$, and $F(q, \dot{q}, u)$ given predictions from the system is termed system identification. For analytically specified mechanical systems, it's often formulated as a linear least-squares problem that can be solved to global optimality in closed form (An et al., 1985). Gradient-based optimization is used when this is not possible.

2.3 Model-Based Control

Given a model specifying a system's dynamics, many techniques from control theory can force a system to follow a specified trajectory, or to track some set point (Liberzon, 2012). Since we limit the scope of experiments in this work to smooth dynamical systems, we use Direct Collocation (DIRCOL) trajectory optimization (Kelly, 2017) in order to generate a 'nominal trajectory' $(\bar{x}_{1:t_H}, \bar{u}_{1:t_H})$ that: (a) transports the system from some initial condition to some desired set point, (b) is dynamically feasible according to the model, and (c) minimizes some cost function defined over the trajectory, which is typically quadratic in the control effort and tracking error.

To track the nominal trajectory in real-time, we use a Time-Varying Linear Quadratic Regulator (TVLQR) (Anderson, 2007). Here, the system dynamics are linearized along the nominal trajectory from DIRCOL, and feedback gains K_t are found using dynamic programming to minimize a quadratic cost function. The actuator input to the system at some time t is:

$$u_t = \pi(x_t) = \bar{u}_t - K_t(x_t - \bar{x}_t) \tag{3}$$

where $\pi : \mathbb{R}^N \to \mathbb{R}^M$ is referred to as the *synthesized policy*.

2.4 Model-Based Reinforcement Learning

Assume we have a finite-horizon Markov decision process $\mathcal{M} = \{\mathcal{X}, \mathcal{U}, \mathcal{T}, R, H\}$, where $\mathcal{X}$ is the state-space, $\mathcal{U}$ is the action-space, $\mathcal{T} : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the system dynamics-model, $R : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the reward function, and H is the horizon-length. In our context, $\mathcal{X} \in \mathbb{R}^{2N}$ is the space of all generalized coordinates and velocities, and $\mathcal{U} \in \mathbb{R}^M$ is the space of all actuator inputs. Additionally, while we do not assume that $\mathcal{T}(x, u)$ is known *a priori*, we restrict our attention to settings in which the reward function $R(x, u)$ is known.

As discussed in Section 2.3, if $\mathcal{T}(x, u)$ were known, we could use DIRCOL to synthesize a trajectory that optimally solves our task, and a *policy* $\pi : \mathcal{X} \to \mathcal{U}$ that tracks this trajectory using TVLQR. If $\mathcal{T}(x, u)$ is not known, it can be learned from data observed while interacting with the environment.

Figure 2 shows the architecture for model-based reinforcement learning used in this work. Polydoros & Nalpantidis (2017) provide a survey of a variety of other approaches. Initially, an episode of training data is generated by randomly actuating the system and observing its state-transitions. A model, $\hat{\mathcal{T}}(x, u)$ is learned from the dataset. The model is then passed through DIRCOL and TVLQR to generate a policy $\pi(x)$ that, if the model were accurate, would solve the task. Naturally, since the model is imperfect, this would not solve the task, but visit novel states.

By repeating this process, novel data is added to the dataset, allowing the learned model to approach the true model in accuracy. Once its prediction accuracy is close enough to the true model, the synthesized policy $\pi(x)$ will be able to solve the task on the real system. However, it is possible for this approach to fail due to a lack of *exploration*. That is, if we attempt to *greedily* solve the task with our model at every episode, the policies being followed may never visit states that are required in the dataset in order to learn a sufficiently accurate model.

A simple approach to exploration in this setup is to add *exploration noise* that adds variance to the policies generated by DIRCOL+TVLQR. Here, we add random perturbations to the nominal trajectory synthesized by DIRCOL before passing it to TVLQR. By doing so, the system is encouraged to visit states in the neighborhood of what is otherwise believed to be an optimal trajectory.

3 Methodology

In its most general form, we model a physical system by modeling: (1) its Lagrangian, and (2) the generalized forces that act on it.

We first present a methodology for modeling the positive-definite mass-matrix $\mathbf{M}_\theta(q)$ using a neural network with parameters contained in θ. We predict $\mathbf{M}_\theta(q) \succ 0$ by predicting the $\frac{N^2+N}{2}$ elements of its Cholesky factor, which is a lower-triangular matrix $\mathbf{L}_\theta(q)$, as is done by Lutter et al. (2019); Haarnoja et al. (2016). Here, the neural network first outputs a vector, of which the first N elements are used as the diagonal of $\mathbf{L}_\theta(q)$, and the remaining $\frac{N^2-N}{2}$ are used for the off-diagonal elements of the lower half of the matrix. We additionally add a constant offset to the diagonals of $\mathbf{L}_\theta(q)$ so that $\mathbf{M}_\theta(q)$ is diagonally dominant and easily invertible given random initializations of θ. We then predict $\mathbf{M}_\theta(q) = \mathbf{L}_\theta(q)\mathbf{L}_\theta^\top(q)$. Additionally, we predict the potential energy, $V_\theta(q)$, using a neural network that maps $q \in \mathbb{R}^N \to \mathbb{R}$, and in the most general case, the generalized forces as a function $F_\theta(q, \dot{q}, u)$ that maps $\mathbb{R}^{2N+M} \to \mathbb{R}^N$. Since we are required to take gradients of $\mathbf{M}_\theta(q)$ as well as $V_\theta(q)$, with restpect to q, in order to compute the system's acceleration, and occasionally Hessians of the Lagrangian in order to measure properties of the model such as local generalized stiffness, we require the non-linearities in the neural network to be at least twice-differentiable. In this work, we use the hyperbolic tangent function (tanh) as the non-linearity.

We have thus far presented the parameterization for the most generic form of a mechanical system. If any prior knowledge is available to a practitioner, they may substitute $\mathbf{M}_\theta(q)$, $V_\theta(q)$, or $F_\theta(q, \dot{q}, u)$ with a more restricted function class than a neural network, so long as as $\mathbf{M}_\theta(q)$ and $V_\theta(q)$ remain twice-differentiable. For example, one may wish to model the system as control-affine with viscous joint-damping. This can be achieved by substi-

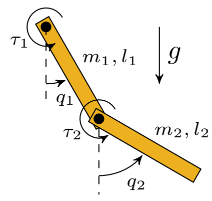

Figure 3: The actuated double pendulum system.

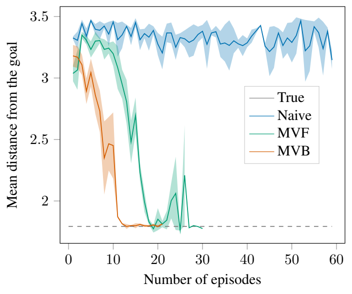

Figure 4: Model-based reinforcement learning performance of various model parameterizations. Here, performance is the mean Euclidean distance between the end-effector and the target over the trajectory, where a lower score is better.

tuting $F_\theta(q, \dot{q}, u) = \mathbf{B}_\theta(q) \cdot u + \eta_\theta(q) \circ \dot{q}$. Here, $\mathbf{B}_\theta(q) \in \mathbb{R}^{N \times M}$ and $\eta_\theta(q) \in \mathbb{R}^N$. Although, bias from this parameterization may not allow certain phenomenon to be accurately modeled.

If $\mathbf{M}_\theta(q)$, $V(q)$ and $F(q, \dot{q}, u)$ are all substituted with analytic models derived from prior knowledge, then this model reduces to a white-box model. On the other hand, if left in its most general form, then the model is still as expressive as a black-box model.

3.1 Parameter Optimization

Having described the parameterization for a model of an arbitrary mechanical system, we now describe how we optimize these parameters to learn a model of a system from data. Given some dataset of a system's state-transitions, $\mathcal{D} = \{q_k, \dot{q}_k, u_k, q'_k, \dot{q}'_k \mid k \in \{1, \dots N\}\}$.

We optimize the parameters θ by minimizing the *prediction loss*:

$$L(\theta) = \frac{1}{N} \sum_{k=1}^{N} \|q'_k - \hat{q}'_k(\theta)\|^2 + \lambda \|\dot{q}'_k - \hat{\dot{q}}'_k(\theta)\|^2 \tag{4}$$

where $\hat{q}'_k, \hat{\dot{q}}'_k$ are predicted using RK4 applied to the model dynamics with inputs $q_k, \dot{q}_k, u_k$ and parameters θ. We then minimize this loss using gradient descent methods such as Adam (Kingma & Ba, 2014).

4 Experiments

In our experiments, we aim to justify the claim that learning dynamics models with our methodology enables more data-efficient model-based reinforcement learning. To do so, we perform model-based reinforcement-learning to achieve the task of inverting a fully-actuated double-pendulum. We compare our method with the use of naive black-box model to parameterize the modeled dynamics.

4.1 Experimental Domain

We focus on the fully actuated double pendulum (shown in Figure 3) as our system of interest, which has highly non-linear dynamics that are chaotic when unforced. As shown in the figure, the system is specified by the masses m_1 and m_2 and lengths l_1 and l_2 of the rods, the gravitational acceleration g, and coefficients that specify the joint torques τ_1 and τ_2, which will be introduced shortly. The system has two generalized coordinates, $q_1 \in [-\pi, \pi)$ and $q_2 \in [-\pi, \pi)$, which correspond to the relative deflection of each rod in radians, as well as control inputs u_1 and u_2. The dynamics of the system are computed by specifying the mass-matrix, $\mathbf{M}_{sys}(q)$, the potential energy, $V_{sys}(q)$, and the generalized coordinates. The specifications for $\mathbf{M}_{sys}(q)$ and $V_{sys}(q)$ can be found in Appendix D. The generalized forces acting on the system are composed of a control torque, as well as viscous joint-damping:

$$F_{sys}(q, \dot{q}, u) = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix} \circ \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix} \tag{5}$$

Here, b_1 and b_2 are the coefficients of the control-matrix $\mathbf{B}_{sys}(q)$, and η_{sys} specify the linear joint-damping coefficients.

4.2 Model-based reinforcement learning

In this experiment, we compare the naive model with two models encoding different amounts of prior knowledge, in their abilities to learn the dynamics of the actuated double pendulum for the purpose of solving a swing up task.

The task is to actuate the system in a manner that brings it to an state that is upright and stationary, i.e. $(q_1, q_2) = (\pi, 0.0)$ $[\mathrm{rad}]$ and $(\dot{q}_1, \dot{q}_2) = (0.0, 0.0)$ $[\mathrm{rad\,s^{-1}}]$, in 2.06 seconds, and then remain stable in this configuration for an additional 0.5 seconds. In order to do so, the model must learn from the available data well enough to accurately generalize predicted dynamics to the relevant parts of the state space. If the model is inaccurate, we expect DIRCOL to produce nominal trajectories that are less dynamically feasible, and TVLQR to produce less accurate linearizations of the dynamics about those trajectories. These two effects will consequently cause the algorithm to require many more interactions with the environment in order to solve the task, when using that model to plan a trajectory.

In this experiment, we refer to naive black-box approach in which $\ddot{q} = NN(q, \dot{q}, u; \theta)$, where $NN(\cdot; \theta) : \mathbb{R}^{2N+M} \to \mathbb{R}^N$ is a feedforward neural network, as the *Naive model*. We compare this model with a model in which $\mathbf{M}_\theta(q)$, $V_\theta(q)$, $F_\theta(q, \dot{q}, u)$ are all represented by neural networks in the manner introduced in Section 3. We refer to this model as the *MVF model*. Here, we note that this model is as expressive as the naive model and that no prior knowledge is encoded aside from the assumption that the system conforms to Lagrangian dynamics. Additionally, we compare a model in which we correctly model the generalized forces acting on the system to be control-affine with linear joint-damping, as follows:

$$F_\theta(q, \dot{q}, u) = \mathbf{B}_\theta(q)u + \eta_{wb} \circ \dot{q} \tag{6}$$

Here, $\mathbf{B}_\theta(q)$ is represented by a neural network, while η_{wb} consists of two learned scalars $[\hat{\eta}_1, \hat{\eta}_2]$. We refer to this model as the *MVB model*. In this experiment, we expect to see that the naive model requires the largest number of interactions to solve the task, followed by MVF, and then MVB.

We follow the procedure for model-based reinforcement learning described in Section 2.4. All trajectories start from the downward equilibrium $(q_1, q_2) = (0.0, 0.0)$ and $(\dot{q}_1, \dot{q}_2) = (0.0, 0.0)$. We provide all models the same initial trajectory found by randomly actuating the system with actions $u_t \sim \mathcal{N}(0, 120^2)$. All trajectories are simulated with $\Delta t = 0.01$ s, and where the action u_t applied is clipped to have a maximum absolute value of 120. We then train all models on this dataset for 5000 epochs, using the Adam optimizer with learning rate of 3×10^{-4}. Every time new data is added to a model's dataset, the model is trained for an additional 1000 epochs.

Next, we perform trajectory optimization using the trained models to get nominal trajectories and a TVLQR tracking controller to synthesize a policy for each model. However, in order to encourage exploration, we add noise sampled from $\mathcal{N}(0, 0.25^2)$ to the nominal trajectories independently to each $q_{1,t}, q_{2,t}, \dot{q}_{1,t}, \dot{q}_{2,t}$ and u_t. Each model's policy is then rolled out on the system, and the trajectory generated is added to each model's dataset. This process is repeated until the trajectories followed by using a given model are reliably solving the task. We repeat this test using three different random seeds.

We measure the performance of the system by measuring the mean Euclidean distance between the end effector and its desired set-point (vertical and stable), over the trajectory. A well-performing model will have a low score using this metric. When evaluating performance, we do not add noise to the nominal trajectories.

Figure 4 shows the results of this experiment. As we can see, the results match our expectations: the MVB parameterization, which incorporates some prior knowledge about the structure of the generalized forces, solves the task with the fewest number of interactions with the environment. Additionally, the MVF parameterization, which incorporates no prior knowledge, takes more interactions to solve the task, but does so eventually. The naive black-box parameterization does not succeed in solving the task in the allowed time-frame. Observing the nominal trajectories and the policies' abilities to follow the trajectories on the real system, we see the behavior we expect for MVF and MVB. The nominal trajectories eventually start looking more realistic, and the policies start doing a better job of following them. However, when using the naive parameterization, DIRCOL appears to be unable to find a solution that transports the system to the desired goal state. Consequently, the trajectories added to the dataset never explore the relevant parts of state-space, resulting in a model that is never able to generalize well enough to solve the task.

Seeing that even the MVF model, which incorporates no prior knowledge, is able to solve the task in a reasonable amount of time, as well as the fact that prior knowledge improves sample efficiency, validates the hypothesis that using the method we presented enables more efficient model-based reinforcement learning than if a naive black-box parameterization of the dynamics were used.

5 Conclusions

We presented a method for parameterizing an arbitrary mechanical system using neural networks, as well as a method for training such models from data. Unlike naive black-box approaches that predict accelerations directly, we use neural networks to parameterize the Lagrangian of a system and the generalized forces that act on it. We showed that such a modular parameterization is flexible enough to allow us to seamlessly incorporate prior knowledge where it is available, allowing a practitioner to precisely balance model bias and variance. We showed on a simulated actuated double pendulum that, even in the absence of prior knowledge, our method learns the dynamics of a complex mechanical system through reinforcement learning more efficiently than a naive, black-box approach. An extended version of this work can be found at `https://arxiv.org/abs/1902.08705`.

ACKNOWLEDGEMENTS

We are thankful to Jeannette Bohg for advice. This work is supported in part by DARPA under agreement number D17AP00032. The content is solely the responsibility of the authors and does not necessarily represent the official views of DARPA.

REFERENCES

Mohamadreza Ahmadi, Ufuk Topcu, and Clarence Rowley. Control-oriented learning of lagrangian and hamiltonian systems. In *American Control Conference (ACC)*, pp. 520–525. IEEE, 2018.

Chae H An, Christopher G Atkeson, and John M Hollerbach. Estimation of inertial parameters of rigid body links of manipulators. In *IEEE Conference on Decision and Control*, pp. 990–995, 1985.

Brian D. O. Anderson. *Optimal Control: Linear Quadratic Methods (Dover Books on Engineering)*. Dover Publications, 2007. ISBN 9780486457666.

K J Åström and P Eykhoff. System identification—a survey. *Automatica*, 7(2):123–162, March 1971.

Sheng Chen, SA Billings, and PM Grant. Non-linear system identification using neural networks. *International Journal of Control*, 51(6):1191–1214, 1990.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop KF: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4376–4384, 2016.

M. Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4): 849–904, 2017. doi: 10.1137/16M1062569.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Daniel Liberzon. *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton University Press, Jan 2012. ISBN 0691151873.

Michael Lutter, Christian Ritter, and Jan Peters. Deep Lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BklHpjCqKm`.

Richard M Murray, Zexiang Li, and S. Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 1994.

Duy Nguyen-Tuong and Jan Peters. Using model knowledge for learning inverse dynamics. In *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2677–2682, 2010.

Athanasios S. Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, May 2017. ISSN 1573-0409. doi: 10.1007/s10846-017-0468-y.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

Nathan Ratliff, Franziska Meier, Daniel Kappler, and Stefan Schaal. Doomed: Direct online optimization of modeling errors in dynamics. *Big Data*, 4(4):253–268, 2016.

Paul J. Werbos, Thomas McAvoy, and Ted Su. Neural networks, system identification, and control in the chemical process industries. In David A. White and Donald A. Sorge (eds.), *Handbook of Intelligent Control Neural, Fuzzy, and Adaptive Approaches*, chapter 10. Van Nostrand Reinhold, New York, 1992.

A Lagrangian Mechanics

For any mechanical system, the *kinetic energy* can be written as:

$$T(q,\dot{q}) = \frac{1}{2}\dot{q}^T\mathbf{M}(q)\dot{q} \tag{7}$$

where $\mathbf{M}(q)$ is positive definite and called the *mass matrix* of the system. Furthermore, the *potential energy* of the system can be defined as a scalar function $V(q)$. Together, these energies specify the *Lagrangian* of a rigid body system as:

$$\mathcal{L}(q,\dot{q}) = T(q,\dot{q}) - V(q) \tag{8}$$

B Efficiently Computing Coriolis Forces

In this section, we describe a method for computing $\mathbf{C}(q,\dot{q})\dot{q}$ in $\mathcal{O}(N^2)$. We begin by stating an expression for the Coriolis matrix in terms of the mass-matrix:

$$\mathbf{C}_{ij}(q,\dot{q}) = \frac{1}{2}\sum_{k=1}^{N}\left(\frac{\partial\mathbf{M}_{ij}}{\partial q_k} + \frac{\partial\mathbf{M}_{ik}}{\partial q_j} - \frac{\partial\mathbf{M}_{kj}}{\partial q_i}\right)\dot{q}_k \tag{9}$$

We can find the the i^{th} element of $C(q,\dot{q})\dot{q}$ as follows:

$$\begin{aligned}\{\mathbf{C}(q,\dot{q})\dot{q}\}_i &= \sum_{j=1}^{N}\left(\frac{1}{2}\sum_{k=1}^{N}\left(\frac{\partial\mathbf{M}_{ij}}{\partial q_k} + \frac{\partial\mathbf{M}_{ik}}{\partial q_j} - \frac{\partial\mathbf{M}_{kj}}{\partial q_i}\right)\dot{q}_k\right)\dot{q}_j \\ &= \underbrace{\frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial\mathbf{M}_{ij}}{\partial q_k}\dot{q}_k\dot{q}_j}_{\text{S1}} + \underbrace{\frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial\mathbf{M}_{ik}}{\partial q_j}\dot{q}_k\dot{q}_j}_{\text{S2}} - \frac{1}{2}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{\partial\mathbf{M}_{kj}}{\partial q_i}\dot{q}_k\dot{q}_j\end{aligned} \tag{10}$$

Note the symmetry between S1 and S2. Combining and moving gradients outside the summation, we get:

$$\{\mathbf{C}(q,\dot{q})\dot{q}\}_i = \sum_{j=1}^{N}\frac{\partial}{\partial q_j}\sum_{k=1}^{N}\mathbf{M}_{ik}\dot{q}_k\dot{q}_j - \frac{\partial}{\partial q_i}\sum_{j=1}^{N}\sum_{k=1}^{N}\frac{1}{2}\mathbf{M}_{kj}\dot{q}_k\dot{q}_j \tag{11}$$

Which can be concisely written as:

$$\mathbf{C}(q,\dot{q})\dot{q} = \nabla_q\left(\mathbf{M}(q)\dot{q}\right)\dot{q} - \nabla_q\left(\frac{1}{2}\dot{q}^\top\mathbf{M}(q)\dot{q}\right) \tag{12}$$

The most expensive operation is the computation of the Jacobian matrix $\nabla_q(\mathbf{M}(q)\dot{q})$, which is computed in $\mathcal{O}(N^2)$.

C Prediction with RK4

Given a the system's generalized coordinates q_t and generalized velocity $\dot{q}_t$ at some time t, we wish to predict $q_{t'}$ and $\dot{q}_{t'}$ at some time $t' = t + \Delta t$. First, we define $x_t = [q_t, \dot{q}_t]^\top$, and consequently, $\dot{x}_t = [\dot{q}_t, \ddot{q}_t]^\top$. Here, $\ddot{q}_t$ is computed according to:

$$\ddot{q} = \mathbf{M}^{-1}(q)\left[F(q,\dot{q},u) - \mathbf{C}(q,\dot{q})\dot{q} - G(q)\right] \tag{13}$$

To integrate a function $g(x,u)$, RK4 is a standard integration scheme:

$$\begin{aligned}k_1 &= \Delta t\cdot g(x_t,u_t) \\ k_2 &= \Delta t\cdot g(x_t + k_1/2, u_t) \\ k_3 &= \Delta t\cdot g(x_t + k_2/2, u_t) \\ k_4 &= \Delta t\cdot g(x_t + k_3, u_t) \\ x_{t'} &= x_t + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\end{aligned} \tag{14}$$

Here, we let $g([q,\dot{q}]^\top, u) = [\dot{q}, \ddot{q}(q,\dot{q},u)]^\top$.

D EXPERIMENTS

Here we specify the double pendulum dynamics in our experiments, as well as parametrizations of the naive black-box model and structured black box models used in our experiments.

D.1 DOUBLE PENDULUM

The mass matrix $\mathbf{M}_{sys}(q)$ and potential energy $V_{sys}(q)$ of the double pendulum are analytically specified below.

$$\mathbf{M}_{sys}(q) = \begin{bmatrix} I_{11} & I_{12} \\ I_{12} & I_2 \end{bmatrix} \tag{15}$$

where

$$I_1 = \frac{1}{3} m_1 l_1^2 \tag{16}$$

$$I_2 = \frac{1}{3} m_2 l_2^2 \tag{17}$$

$$I_{11} = I_1 + I_2 + m_2 {l_1}^2 + m_2 l_1 l_2 \cos q_2 \tag{18}$$

$$I_{12} = I_2 + \frac{1}{2} m_2 l_1 l_2 \cos q_2 \tag{19}$$

$$V_{sys}(q) = -\frac{1}{2} m_1 g l_1 \cos q_1 - m_2 g \left(l_1 \cos q_1 + \frac{l_2}{2} \cos(q_1 + q_2) \right) \tag{20}$$

For the experiments, the parameters specified in Table 1 are used to define the system dynamics.

Parameter	Value	Unit	Parameter	Value	Unit
m_1	10.0	kg	b_1	1.0	$\mathrm{N\,m}$
m_2	10.0	kg	b_2	1.0	$\mathrm{N\,m}$
l_1	1.0	m	η_1	-0.5	$\mathrm{N\,m\,s\,rad^{-1}}$
l_2	1.0	m	η_2	-0.5	$\mathrm{N\,m\,s\,rad^{-1}}$
g	10.0	$\mathrm{m/s^2}$			

Table 1: Parameters specifying the system dynamics for both experiments.

D.2 MODEL PARAMETERIZATION

The Naive model consists of a multi-layer perceptron with 3 hidden layers of dimension 64. The MVF model consists of a multi-layer perceptron for $\mathbf{M}(q)$, $V(q)$ and $F(q, \dot{q}, u)$ each, with 3 hidden layers of dimension 32. These network sizes are chosen so that the naive and MVF model have roughly the same number of trainable parameters. All other models that use neural networks to parameterize components use neural networks with with 3 hidden layers of dimension 32 for those components.