UNIVERSITY OF CALIFORNIA

Los Angeles

Data-efficient Deep Learning of Dynamical Systems

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Tianyi Wang

2023

ABSTRACT OF THE DISSERTATION

Data-efficient Deep Learning of Dynamical Systems

by

Tianyi Wang
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2023
Professor Vwani P. Roychowdhury, Chair

The synergy between dynamical systems and deep learning (DL) has become an increasingly popular research topic because of the limitation of classic methods and the great potential of DL in addressing these challenges through the data-fitting and feature-extraction power of deep neural networks (DNNs). DNNs have demonstrated their ability to approximate highly complicated functions while enjoying good trainability, which can help dynamical system modeling with both the search of solution and the expressiveness of models. Furthermore, the feature extraction ability of DNNs have proven useful in identifying system states identification when the state cannot be defined from first-principles.

Conversely, the study of dynamical systems has benefited DL. Viewing DNNs as discretization of ordinary differential equations (ODEs) inspires a novel family of models named neural differential equations which offer unique advantages in time series learning, especially the modeling of dynamical systems. From another perspective, viewing the optimization of deep learning models as a dynamical system on the loss landscape enables better analysis and enhancement of the optimization processes.

This work focuses on this interplay. We develop novel deep learning methods to efficiently model dynamical systems, incorporating physical prior knowledge and meta-learning techniques. By analyzing the dynamics of the optimization process, we also design a novel variant

of stochastic gradient descent to enhance the resilience of DNNs against weight perturbations, enabling their deployment on analog in-memory computing platforms where analog noise is inevitable. Through these investigations, we contribute to the growing body of research on the intersection of dynamical systems and deep learning, paving the way for innovative solutions to complex real-world problems.

The dissertation of Tianyi Wang is approved.

Lieven Vandenberghe

Jonathan Chau-Yan Kao

Louis-Serge Bouchard

Vwani P. Roychowdhury, Committee Chair

University of California, Los Angeles

2023

To my family . . .

CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to express my sincerest gratitude to my advisor, Professor Vwani Roychowdhury. Always intellectually vigorous, inspiring, yet patient and supportive, he has provided me invaluable help and guidance. I will always remember the "aha moments" we shared in his office, some in the day, some in late nights.

I also would like to acknowledge the members of my doctoral committee, Professor Lieven Vandenberghe, Professor Jonathan Kao and Professor Louis Bouchard for taking the time to serve on my doctoral committee, read my thesis, and provide advice on my work.

It has been a fortunate journey, through which I received friendships that can last for life. I am glad to have met many good friends in our lab. We shared sparks of thoughts, supports in our lives and careers, and many late night tacos. I also want to extend my heartfelt thanks to all of my friends, who care for me and deeply understand me.

I thank all my collaborators, in particular, Dr. Qiaofeng Li and Dr. Zhe Wan. This thesis would not be possible without them working with me. Innovative and professional, they set great models for me during our collaboration.

Most importantly, I thank my parents and my family. It is their love for me and my love for them that makes me who I am.

VITA

2012–2016	B.S. (Physics), Fudan University, Shanghai, China.
2016–2018	M.S. (Electrical Engineering), UCLA, Los Angeles, California.
2018–2023	Ph.D. Candidate (Electrical and Computer Engineering), UCLA, Los Angeles, California.

CHAPTER 1

Introduction

1.1 Background

To understand, predict for, and further to control a system, the dynamics of its evolution needs to be modeled. Concretely, given the observed behaviors of a system, the goal is to derive succint models that can replicate the systems behaviors under arbitrary initial / boundary conditions and system parameter settings.

Traditionally, dynamical systems have been modeled by formulating analytical partial differential equations (PDEs) or ordinary differential equations (ODEs) that govern carefully chosen state variables, based on comprehensive knowledge of the systems being studied. In this approach, observed data are used solely to estimate the coefficients of the equation terms. However, this approach is faced with challenges such as high computational cost and limited model capacity. Even when the first principles are well established, dynamics modeling is hard to scale to complex, large-scale, non-linear or noisy systems due to intractable computational demands. For novel or complex systems, there are hardly universal and succinct first principles, rendering their modeling even more complicated.

Advances in deep learning (DL) during the past decade have given rise to the new paradigm of data-driven modeling of dynamical systems. Deep neural networks (DNNs) have been used in various aspects of the modeling, ranging from state estimation [LNB18, NG20], system identification and learning of dynamics [RBP17, LRP18, CRB18, CLK19, GDY19, LKA20, HHM22, PBJ22], efficient solving of the dynamical system [RPK19, LZK21], to even the extraction of states from data [LKB18, ZWT18, XLS19, CLK19, CHR22]. These works adopt the data-driven approach by replacing one or more components of the traditional

modeling framework with deep neural networks.

On one hand, data-driven modeling offers the possibility for superior computation efficiency compared to classic methods [HLM17, FWY20, HMC21]. On the other hand, it enables advances towards automated rule discovery and learning of the dynamics of a system from observations [CGV19, RCD19, CLK19, LKA20, IMW20, LT21].

In general, the goal is to maximize the generality of the data-driven models while minimizing the required system-specific expert knowledge. These works have demonstrated their usage on datasets generated by various systems, and the design of data-driven approaches for different practical scenarios is a topic of great continuing interest.

However, deep learning is data-hungry. DNNs require a large amount of data to reach reasonable performance, and would not guarantee to extrapolate well for unseen scenarios outside of the training data distribution. Unconstrained neural networks, trained to directly fit time-series data generated by dynamical systems, often cannot replicate dynamics under unseen initial or boundary conditions, whereas the amount of data required to improve such poor generalizability is usually prohibitively expensive.

Recent studies have shown that incorporating physics constraints into deep learning is vital for models' generalization ability, learning efficiency, and interpretability [ZSZ18, AGS21]. For example, Hamiltonian Neural Network [GDY19] and Lagrangian Neural Network [LRP18, CGH20] introduced the physical prior of energy conservation to neural networks for learning dynamics. Geometric deep learning [AGS21, BBC21] and equivariant neural networks [WGW18, TSK18, FWW21, SHW21] leverage geometric symmetry properties of the network input to improve the quality of inherent knowledge learned by neural networks. Successful applications include drug discovery [GHS16, JGW21], quantum chemistry [GSR17], and particle physics [KMT19, QG20].

1.2 Contributions and Dissertation Overview

In this dissertation, we address the interplay between deep learning and dynamical systems and make our three main contributions in the following chapters:

Chapter 2 By analyzing the optimization dynamics of deep neural networks, we propose Hessian-aware (HA-SGD), a novel gradient descent algorithm tailored to seek flat minima within the network's loss landscape. This optimization algorithm, which simultaneously minimizes both the loss value and landscape sharpness, can crucially bolster the noise resiliency of neural network models deployed on analog compute-in-memory (CIM) platforms.

Chapter 3 To enable data-efficient modeling of dynamical systems that can generalize well to unseen initial conditions and boundary conditions, we use the reduced-order modeling of Slinkies as a case study and incorporate physics prior knowledge into the neural model to achieve accurate dynamics modeling with high data efficiency and great generalization ability.

Chapter 4 The data-efficiency of dynamical system modeling can be further boosted with the knowledge of the functional form of the dynamics, which is usually not available in the data-driven modeling scenarios. We propose **i**nterpretable **M**eta **N**eural **ODE** (iMODE) that uses meta-learning to learn the common dynamics form shared by multiple dynamical system instances without requiring labels of their varied physical parameters. We demonstrate the superior modeling quality and efficiency of iMODE with various dynamical systems. In addition, the adaptation parameters in iMODE are interpretable and can be used to facilitate knowledge discovery for unknown dynamical systems.

CHAPTER 2

Modified Stochastic Gradient Descent Dynamics for Noise-resilient Analog In-memory Computing

2.1 Introduction

Analog in-memory computing emerges as a promising novel technique for deploying deep neural networks (DNNs) in mobile devices and the Internet of Things (IoT) on a large scale, because of its potential of increasing the energy-efficiency of DNNs' inference by orders of magnitude [HM13, GBB17].

However, analog noise is inevitable in analog devices, whose impact to deep neural network inference requires further study. Based on the charge-trap-transistors (CTTs) analog neural network implementation proposed by UCLA CHIPS (Center for Heterogeneous Integration and Performance Scaling), we develop a simulation framework that simulates analog implementations of common DNN architectures including various Convolutional Neural Networks (CNNs) and Multilayer Perceptrons (MLPs).

The simulations show that analog noise will lead to catastrophic performance degradation of DNNs, rendering it impractical to deploy large DNNs to analog platforms. To improve the analog resiliency of the DNNs at the software level, where no changes to the hardware and device are needed, we introduce a Hessian-Aware Stochastic Gradient Descent (HA-SGD) algorithm, which intentionally search for flat minima in the loss landscape of DNNs, resulting in parameter sets that are resilient to analog noise. HA-SGD has proved to be very effective. At 10 % device noise, network trained with injected noise of 10 % achieves a top5 accuracy of 61.67 %, which is more than 4 times the top5 accuracy of the networks trained by standard

SGD (14.37 %).

The following sections in this chapter are organized as follows. Section 2.2 introduces the analog in-memory computing architecture. The simulation framework for the inference of DNNs deployed to analog computing platforms based on the hardware characteristics, and the quantitative analysis of the impact of analog noise is presented in Section 2.3. The HA-SGD algorithm is described in Section 2.4, with the empirical evidence of its effectiveness and the theoretical explanation.

2.2 Analog Compute-in-memory Deep Learning Accelerator

Deep learning based on deep neural networks (DNNs) are now only computed by digital machines [CKE17, JYP17, LKK18], where information and computation are both digitized (e.g. with 32-bit floating point precision) and usually operated within a von Neumann architecture. Although the von Neumann architecture has been prevalent and successful under the rapid development of the Moore's Law, it has encountered the von Neumann bottle neck [Wil17], a limitation of the bandwidth between the processors and the memory, especially for data-intensive computations such as deep learning.

To address this bottleneck, compute-in-memory (CIM) architectures using analog non-volatile memory (ANVM) as the synaptic weight have emerged as the hope to improve energy-efficiency and throughput for DNN operations by several orders of magnitude [HGP19]. Preliminary results have been demonstrated on various analog devices [BBS16, GBB17, LBL18, SLY18, ZZP18, SPH16, GWI19].

In particular, to enable ultra-large-scale analog computing for useful DNNs, our collaborator at the UCLA Center for Heterogeneous Integration and Performance Scaling (UCLA CHIPS) propose to leverage the charge-trap transistor (CTT) along with the mature and scalable commercial CMOS technologies to fabricate neuromorphic engines for neural network *inference* [GWI19, Wan20]. Featuring the compatibility with existing CMOS technologies, a fully-analog hardware architecture that is scalable to a multi-chip system-on-wafer (SoW) is

designed to allow the system to perform all inference operations required by DNNs through analog signal processing locally in the memory, avoiding the cost (in both energy and delay) of analog-to-digital and digital-to-analog conversions in mixed-signal architectures.

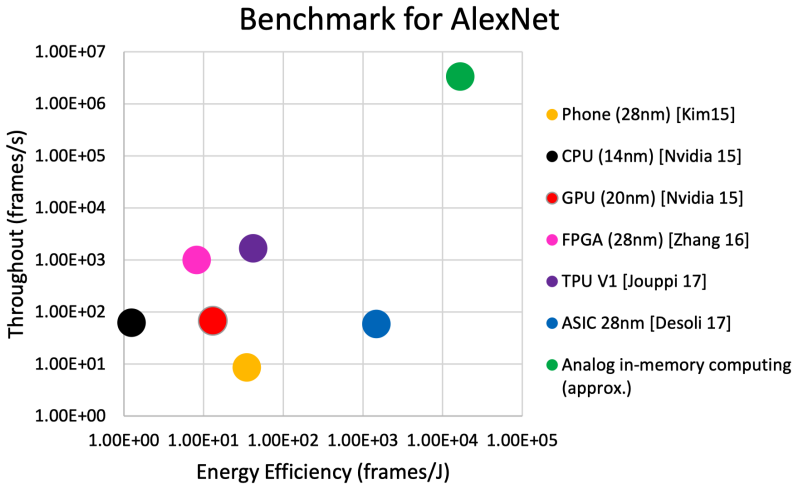


Figure 2.1: (Taken from [Wan20]) The throughput on image recognition application and the efficiency (defined as throughput per Watt) of some typical and novel architectures. The analog in-memory computing has the potential to significantly improve the throughput and efficiency compared to digital architectures.

2.2.1 Parameter Programming Noise in Analog CIM Devices

In digital systems, the precision is guarded by sohisticated digital design flows and error-correction schemes. However, due to the nature of analog devices, analog computing is approximated computing. Encoding real numbers into continuous physical quantities, the analog memory cannot guarantee perfect fidelity due to limits of physical principles. For the emerging ANVM devices, the programmed continuous analog state (e.g., current under some fixed bias) deviates from the intended value due to several reasons:

1. Programming process. Since the programming mechanism of many emerging ANVM devices has uncertainty, they require a closed-loop verification process to determine whether the programming is finished. The criterion for the termination of the programming process depends on the user's ability to fine-tune the device and the precision of the measurement hardware. The non-ideal termination of the programming will affect

the accuracy of programming.

2. Device and cycle variation. For devices that use an open-loop process for programming, the variation among the devices and among the different cycles of the same device will contribute to the error of the programming.

3. Imperfect data retention. The analog state of the device will keep changing over time due to various reasons related to its programming mechanism. It is important to predict the time of failure for the ANVM-based systems and suggest solutions such as data refreshing. However, the refreshing process itself will again suffer from the programming errors.

As a result, there is ubiquitous continuous error in the programmed states. Figure 2.2 shows that such error can usually be modeled by a Gaussian noise.

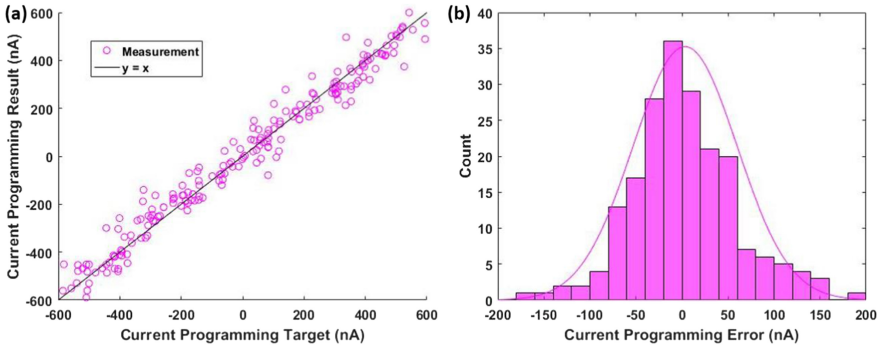


Figure 2.2: Typical ANVM device programming error characteristics (adapted from [GWI19]) in scatter plot (a) and histogram (b).

Indeed, such error could be reduced, but making analog computing as accurate as its digital counterparts using sophisticated circuits would undermine its advantage in energy and area efficiency.

2.2.2 Hardware Characteristics

In this section, we analyze the analog errors in CTT ANVM-based CIM devices in more detail. Figure 2.3 illustrates CTT ANVM-based vector-matrix multiplication. Matrix entries

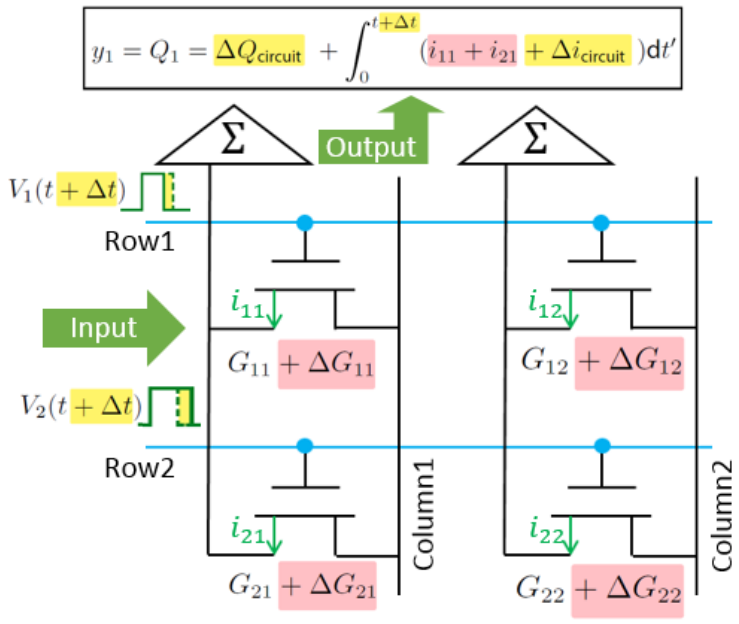


Figure 2.3: **Operation schematic**: array of ANVM such as CTTs can be used for vector-matrix multiplication (VMM). Twin-CTT cells are programmed to represent the synaptic weights with their differential conductance (denoted by G_{ij}). Input is represented by pulse-width modulated (PWM) signal sent to the word-line (WL) of the array, which connects the gates of the CTTs in a row. Multiplication is thus computed by Ohm's law ($i_{ij} = V_i G_{ij}$). Summation is computed by collecting the drain current of all the devices in a bit-line, which consists of two columns due to the twin-CTT cell structure. The equation in the top box further elaborates how y_1, the first among the two output elements, is obtained by integrating the output sum current for a fixed time interval t.
Error analysis: Device related error terms are highlighted in red, and the circuit related error terms are highlighted in yellow.

are programmed into the conductance of CTT cells, the input vector gets multiplied with the matrix through Ohm's law, and the summation is done by Kirchhoff's Current Law.

Programming-error in CCT conductance is inevitable; moreover, the programmed-conductance will relax over time. The actual conductance of an analog-cell can be modelled by $G = G_{\text{target}} + \Delta G_{\text{relax}}(t) + \Delta G_{\text{rand}}$, where G is the actual conductance, G_{target} is the target conductance, ΔG_{rand} is a Gaussian random variable of zero mean reflecting the programming error, and $\Delta G_{\text{relax}}(t)$ reflects the relaxation. With proper over-programming, the two types of error in conductance combined can be modeled with a Gaussian distribution. (see Figure 2.4(a)). We quantify the level of such uncertainty by the ratio between the standard

deviation of the device conductance error and the entire range of device conductance, which is around 5 % to 10 % from our CTT characterization, and can be more for the other analog devices.

In additional to the programming-error of the analog devices, peripheral circuits can contribute to errors in the computation. We note that by using appropriate designs, it is always possible to reduce any circuit noise so that the device noise is the only significant noise during the VMM.

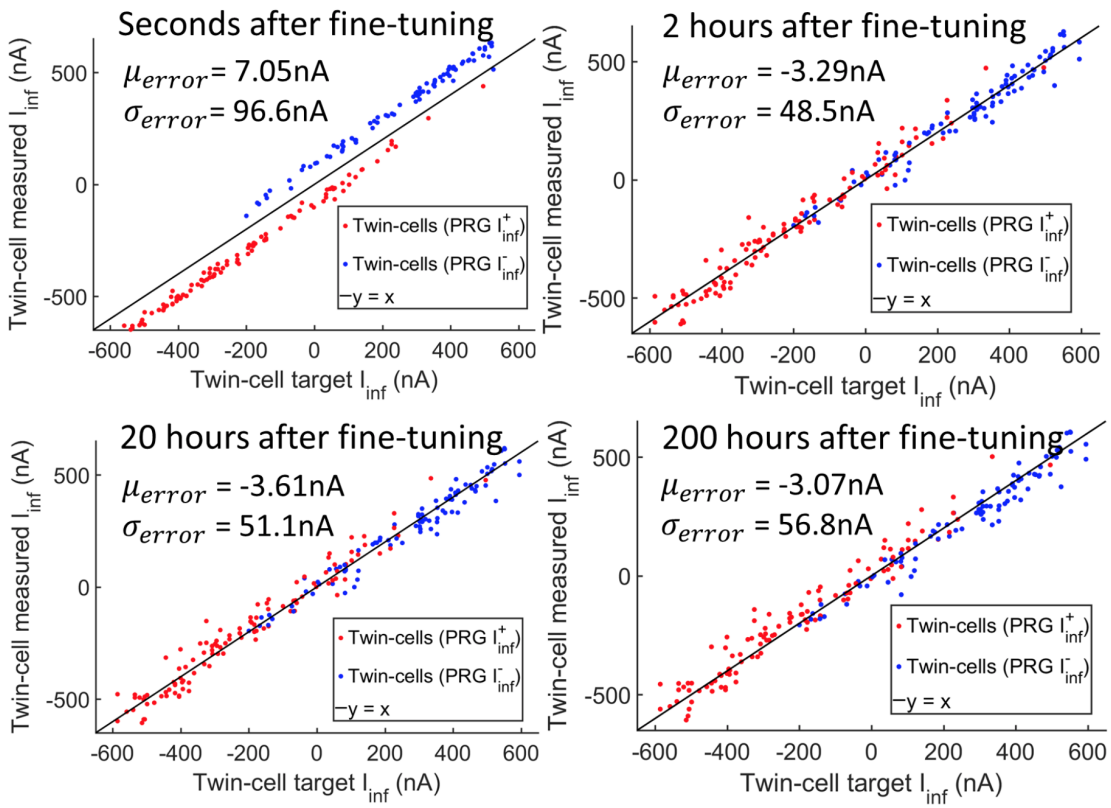
2.3 Performance Degradation of Deep Neural Networks Under Analog Parameter Noise

As analog computing architectures emerge for deep learning acceleration, the analysis of the influence on deep model performance due to analog device errors is necessary. For neural networks deployed to digital systems, the impact of errors in digital memories has been studied [RGP18], but the continuous nature of the analog device error makes it fundamentally different from the discrete error in digital memories. For analog systems, studies on relatively small networks for simple tasks have been reported [MQC18, HGP19], but it is not clear whether larger DNNs for more practical applications are — or can be made — resilient to such uncertainties from analog computation.

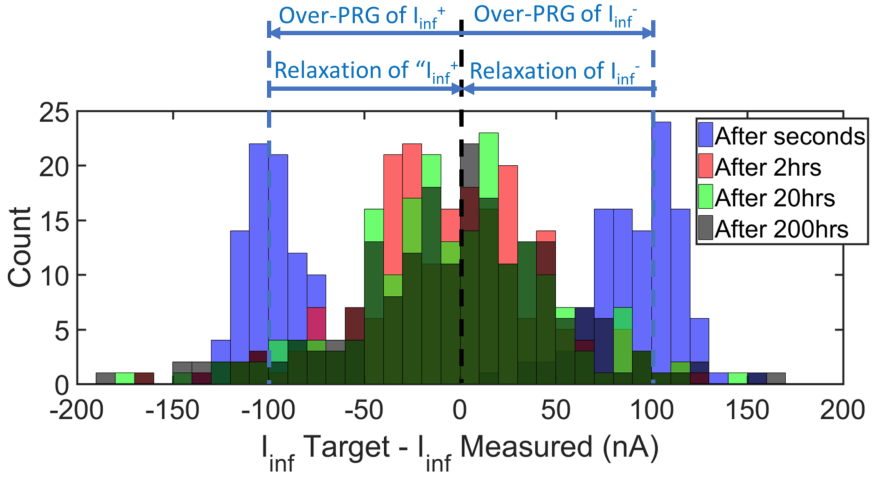
Analog device errors randomly perturbs neural network parameters away from trained optimal values, which degrades the model performance. To quantify the degrading effect in analog CIM architectures, we propose a simulation framework for the analog inference of both multilayer perceptrons (MLPs) and convolution neural networks (CNNs).

2.3.1 Error modeling

Based on the hardware characteristics, we model the uncertainties in analog implementations of DNN components including (1) fully connected layers; (2) convolutional layers; (3) batch normalization layers, and (4) shortcuts in residual networks.



(a) CTT cells' actual conductance v.s. target conductance



(b) CTT cells' conductance error histograms

Figure 2.4: **Uncertainties of CTT ANVM devices**: (a) The figures plot the device conductance measured through current versus the programming target values, seconds, 2 h, 20 h and 200 h after programming. The initial conductance is over-programmed to compensate for the relaxation. The standard deviation of the errors is thus 4 % to 5 % of the dynamic range of the devices, which is −600 nA to 600 nA. (b) The histogram of the fine-tuning error achieved, including the relaxation (at room temperature) after up to 200 hours of the fine-tuning. After 2 h, the error distribution is Gaussian-like.

As is introduced above, DNN's parameters (synaptic weights) $\{w_i\}_{i=1}^n$ are proportionally programmed into the conductance of CTT devices $\{G(w_i)\}_{i=1}^n$. The conversion relation is shared among the CTT devices per die.

To maximize resolution and minimize the influence of noise, the full dynamic range of the devices should be utilized to encode the DNN weights. The uncertainties of the conductance is thus translated to the uncertainties of the DNN parameters by

$$\frac{\sigma_G}{\text{range}_G} = \text{noise level} = \frac{\sigma_{\boldsymbol{w}}}{\text{range}_{\boldsymbol{w}}}$$

where σ_G is the standard deviation of the conductance error in a die, range_G is the dynamic range of the conductance in the die; $\sigma_{\boldsymbol{w}}$ is the standard deviation of the resulted DNN weight error, and $\text{range}_{\boldsymbol{w}}$ is the range of the weights programmed to the die.

2.3.2 Modeling the Analog Implementation of DNN Components

Fully-connected layers

Fully connected layers are the most basic type of layers in DNNs. They (before any non-linear activation) simply perform a linear transformation on its input. The linear transformation is defined by the parameter matrix of the layer, $\mathbf{W}$:

$$\boldsymbol{x}_{\text{out}} = \mathbf{W}\boldsymbol{x}_{\text{in}}$$

With analog noise, each entry W_{ij} in $\mathbf{W}$ is perturbed to $W_{ij} + \nu_{ij}$, $\nu_{ij} \overset{iid}{\sim} p(\nu)$. $p(\nu)$ is the device-dependent random weight error distribution, which can be (naïvely) modeled as a Gaussian distribution with standard deviation $\sigma_{\boldsymbol{w}}$.

There can also be bias terms, $\boldsymbol{b}$, which can be merged into $\mathbf{W}$:

$$\boldsymbol{x}_{\text{out}} = \mathbf{W}\boldsymbol{x}_{\text{in}} + \boldsymbol{b} \Rightarrow \boldsymbol{x}_{\text{out}} = \begin{bmatrix} \mathbf{W} & \boldsymbol{b}/s \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\text{in}} \\ s \end{bmatrix}$$

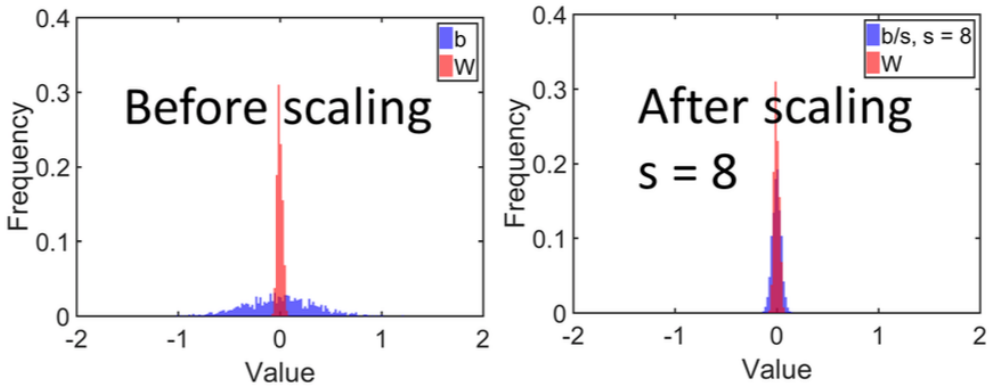


Figure 2.5: **Range-matching - an example**: set $s = 8$ to shrink the range of $\boldsymbol{b}$'s entries down to match that of $\mathbf{W}$'s entries.

Note that there is a degree of freedom of the scaling factor s. The scaling factor can be utilized to match the range of $\mathbf{W}$'s entries and the range of $\boldsymbol{b}$'s entries, so that both of $\mathbf{W}$ and $\boldsymbol{b}$ make use of the full dynamic range of the device and minimize the effect of the device noise (see Figure 2.5). Experiments show that such range-matching significantly improves analog resiliency when the range of $\boldsymbol{b}$ dominates the range of $\mathbf{W}$, which is found to be often the case in practice.

Convolutional layers

Convolutional layers are another prevalent layer type in DNNs, where (before any non-linear activation) a small-size filter ("kernel") is used to linearly transform the input over sliding windows. The kernel is again a matrix, and thus in an analog implementation it will be perturbed in a way similar to that of fully-connected layers.

One analog in-memory computing design that differentiates the convolutional layer case from the fully-connected layer case is "unfolding". In a convolution operation, the same filter weights is reused on multiple positions. Equivalently, to eliminate the time delay and syncing issues, one can duplicate the filter for each position, and use routing between devices to perform the convolution. In contrast, digital implementations will loop and calculate for each position (or perform limited unfolding).

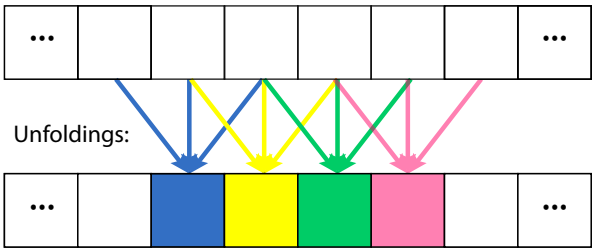


Figure 2.6: **1D example of "unfolding"**: a convolutional filter is duplicated into identical ones for each sliding-window position so that the convolution layer can be implemented in the same way as the fully connected layer is. Each color corresponds to a replica of the filter.

Batch normalization layers

Batch Normalization has been widely employed and shown great success in accelerating and stabilizing the training of DNNs.

During inference, it's nothing but a linear operation, normalizing its input for each channel individually with learned parameters (mean μ_c, variance σ_c^2, learnable scaling factor γ_c and learnable bias β_c, for each channel c):

$$\boldsymbol{x}_{\text{out},c} = \mathbf{W}_{\text{eff},c}\boldsymbol{x}_{\text{in},c} + \boldsymbol{b}_{\text{eff},c}$$

where $\mathbf{W}_{\text{eff},c} = \dfrac{\gamma_c}{\sqrt{\sigma_c^2 + \epsilon}}$, $\boldsymbol{b}_{\text{eff},c} = \beta_c - \dfrac{\gamma_c \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$ and ϵ is a small scalar used to avoid singularity.

Therefore, it can be implemented through convolutional layers with unit-size convolutional kernel, using $\mathbf{W}_{\text{eff},c}$ as weight, $\boldsymbol{b}_{\text{eff},c}$ as bias, and unit stride.

The range-matching also needs to be done for batch normalization layers.

Shortcut in residual networks

Shortcut connections were introduced by ResNet [HZR16] to address the gradient vanishing problem, and have become an indispensable component in deep neural networks.

Without noise, it's a special form of linear operation where $\mathbf{W} = \mathbf{I}$.

In our noise-considering implementation of shortcut connections, we assume that each

positive entry of the identity matrix suffers from a Gaussian noise $\boldsymbol{u}$. $u_i \overset{iid}{\sim} \mathcal{N}(0, \delta)$, where δ is the device noise, and $\mathbf{W} = \mathbf{I}_{\text{shortcut}} = \text{diag}(\boldsymbol{u}) + \mathbf{I}$. This is to reflect the noise from possible circuit implementation of the shortcut (e.g. current mirror).

ReLU

Rectified linear unit (ReLU) and its variants need not to involve CTT, and can be implemented by circuits (e.g. comparators), thus we don't consider noise in activation layers.

Residual block

Using components discussed, the basic building block of residual convolutional neural network — residual block — can be assembled (Figure 2.7).

Assemble and simulate arbitrary architecture

We implemented the basic building blocks mentioned above with analog noise injection API in PyTorch. Currently, two forms of noise distribution – Gaussian and uniform – have been implemented; more distributions can be added. The modules implemented allows the noise level of each component to be individually set dynamically to characterize hardware specs.

With the basic building blocks implemented with proper API, most modern DNN architectures can be assembled and hence modeled and simulated.

2.3.3 Simulations

Our modeling framework is used to simulate DNNs based on the WideResNet architecture [ZK16]. The networks are trained on commercial GPUs using 32-bit floating point precision. However, during testing of the trained network, noise models are included to emulate the behavior of analog devices. Each trained network is instantiated with analog

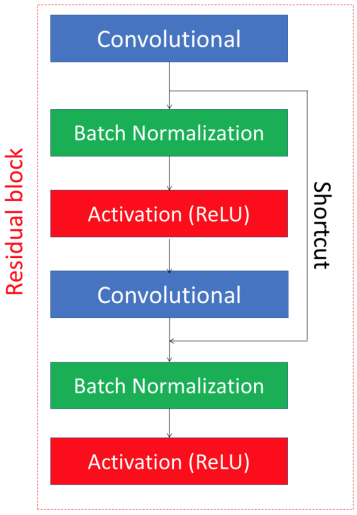
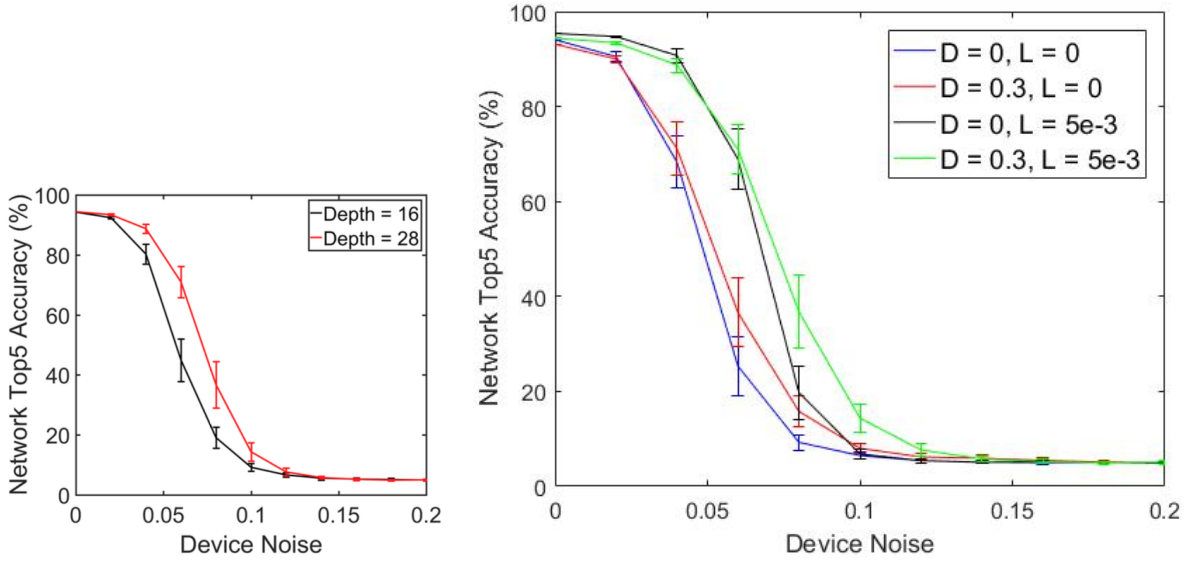


Figure 2.7: Residual Block Structure

errors added for 50 times by independently sampling from the device noise statistics. Then the system is evaluated by all testing patterns on all instantiated neworks to obtain the statistics for network performance (classification accuracy).

In the experiment, two different sizes of WideResNet with depth of 16 (17.1 million weights) and 28 (36.5 million weights) are trained and tested on the CIFAR-100 dataset (same for all network simulation results shown). In Figure 2.8(a), all networks presented show significant degradation of the network accuracy as the device noise increases, while the larger network is more resilient. When device noise is 6 %, the top5 accruacy is degraded from 94.28 % to 44.96 % for the 16-layer network and from 94.39 % to 70.99 % for the 28-layer network. Both networks start to fail completely at device noise of 14 %. For comparison, the 28-layer network structure is used for the later experiments. Some improvement in both the network accuracy and the resiliency can be achieved by using the standard generalization methods such as ℓ_2-regularization and dropout. For example, at 4 % device noise the top5 accuracy can be improved from 68.25 % to 90.72 % as shown in Figure 2.8(b).



(a) The degradation of network due to analog device noise

(b) Effect of conventional regularization methods

Figure 2.8: **Simulation experiment results**: **(a)**: Two WideResNet models of depth 16 and 28 are trained and tested on the CIFAR-100 dataset with optimzied ℓ_2-regularization and dropout factor. **(b)**: Accuracy of WideResNet (depth = 28) is trained with different ℓ_2-regularization strength (L) and dropout factors (D). Some improvement in the analog resiliency can be achieved, but the degradation is still much.

2.4 Hessian-aware Stochastic Gradient Descent (HA-SGD)

Efforts can be put to both hardware and software side to improve an ADNN's accuracy. Hardware precision can be improved at the cost of thoughput, energy and area-efficiency. On the other hand, computation errors due to device noise can be compensated for by better training algorithms that generate more resilient networks.

To improve the analog resiliency of the DNNs at the software level, where no changes to the hardware and device are needed, we introduce a *Hessian-Aware Stochastic Gradient Descent (HA-SGD)* algorithm, motivated by including Hessian information in descent steps to converge to a flat minimum, where the performance degradation caused by perturbations to parameters is low.

2.4.1 ADNN Performance and Minima Flatness

A DNN is trained through optimizing its loss function $J(\boldsymbol{w}; \mathcal{X})$, where $\boldsymbol{w}$ is the parameters of the DNN and $\mathcal{X}$ is the training data.

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} J(\boldsymbol{w}; \mathcal{X})$$

During the inference of a trained DNN, analog CTT noise is adding random perturbation to $\boldsymbol{w}^*$. Since the loss function quantifies the performance of the DNN model, we want the loss function not to go up much after such perturbation. This equivalently means that we want the loss function landscape around $\boldsymbol{w}^*$ to be flat.

The Hessian at local minima captures the curvature information of the loss landscape, and norms of Hessian (e.g. spectral norm, trace) have been used as a measure of flatness [CCS16, KMN16, DPB17]. In the following we show that the performance degradation of DNN due to analog noise can also be quantified using Hessian, and motivate our HA-SGD algorithm.

Consider a random perturbation $\Delta\boldsymbol{w} \sim p_{\Delta w}(\Delta\boldsymbol{w})$, whose distribution $p_{\Delta w}(\Delta\boldsymbol{w})$ depends on device characteristics. We denote its mean as $\boldsymbol{\mu}$ and covariance matrix as $\boldsymbol{\Sigma}$. We assume

$\boldsymbol{\mu} = 0$ because constant shifts can always be compensated for.

Expand the loss function at the perturbed position near a minimum, $J(\boldsymbol{w}^* + \Delta\boldsymbol{w}; \mathcal{X})$, to the second order:

$$\begin{aligned} J(\boldsymbol{w}^* + \Delta\boldsymbol{w}; \mathcal{X}) =& J(\boldsymbol{w}^*; \mathcal{X}) + \boldsymbol{g}(\boldsymbol{w}^*)^\mathsf{T}\Delta\boldsymbol{w} + \\ & \frac{1}{2}\Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w} + \mathcal{O}(\Delta\boldsymbol{w}^3) \end{aligned} \tag{2.1}$$

where $\boldsymbol{g}(\boldsymbol{w}^*) := \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^*; \mathcal{X})$ is the gradient at $\boldsymbol{w}^*$, and $\mathbf{H}(\boldsymbol{w}^*) := \nabla^2_{\boldsymbol{w}} J(\boldsymbol{w}^*; \mathcal{X})$ is the Hessian at $\boldsymbol{w}^*$.

Since $\boldsymbol{w}^*$ is a local minimum, $\boldsymbol{g}(\boldsymbol{w}^*) = 0$, and $\mathbf{H}(\boldsymbol{w}^*)$ is (semi-)positive definite.

Take *expectation* of the Equation (2.1), we have

$$\begin{aligned} \mathbb{E}\left[J(\boldsymbol{w}^* + \Delta\boldsymbol{w}; \mathcal{X})\right] &\approx J(\boldsymbol{w}^*; \mathcal{X}) + \frac{1}{2}\mathbb{E}\left[\Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w}\right] \\ &= J(\boldsymbol{w}^*; \mathcal{X}) + \frac{1}{2}\left[\operatorname{Tr}\left(\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\Sigma}\right) + \boldsymbol{\mu}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\mu}\right] \\ &= J(\boldsymbol{w}^*; \mathcal{X}) + \frac{1}{2}\operatorname{Tr}\left[\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\Sigma}\right] \end{aligned} \tag{2.2}$$

Thus the expected value of the loss function is higher than that at $\boldsymbol{w}^*$ by a gap of $\frac{1}{2}\operatorname{Tr}\left(\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\Sigma}\right)$.

We can also compute the *variance* of the loss function value due to random perturbations:

$$\begin{aligned} & \operatorname{Var}\left[J(\boldsymbol{w}^* + \Delta\boldsymbol{w}; \mathcal{X})\right] \\ \approx\ & \boldsymbol{g}(\boldsymbol{w}^*)^\mathsf{T}\boldsymbol{\Sigma}\boldsymbol{g}(\boldsymbol{w}^*) + \boldsymbol{g}(\boldsymbol{w}^*)^\mathsf{T}\operatorname{Cov}\left[\Delta\boldsymbol{w}, \Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w}\right] + \frac{1}{4}\operatorname{Var}\left[\Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w}\right] \\ =\ & \frac{1}{4}\operatorname{Var}\left[\Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w}\right] \end{aligned} \tag{2.3}$$

For normally distributed $\Delta\boldsymbol{w}$, or for $\Delta\boldsymbol{w}$ that has independent components (that is, $\Delta w_i \perp\!\!\!\perp \Delta w_j$ if $i \neq j$),

$$\frac{1}{4}\operatorname{Var}\left[\Delta\boldsymbol{w}^\mathsf{T}\mathbf{H}(\boldsymbol{w}^*)\Delta\boldsymbol{w}\right] = \frac{1}{2}\operatorname{Tr}\left[\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\Sigma}\mathbf{H}(\boldsymbol{w}^*)\boldsymbol{\Sigma}\right] \tag{2.4}$$

It's desirable to have both the *expectation* and *variance* low, so that the DNN will reliably have low degradation under analog noise.

Thus, low norm (trace norm and Frobenius norm) of the Hessian (which is the sum and the sum of the squares of the eigen values respectively) is desirable at $\boldsymbol{w}^*$, which corresponds to wide valleys in the loss function landscape.

The conventional training techniques of DNNs, including various optimizers and regularization techniques, however, don't regularize the eigenvalues of the Hessian, and thus would not be able to find wide valleys/flat minima. This explains our empirical findings of severe degradation in performance. Attempts have been made to find such wide valleys in [HS97, CCS16], but such attempts have remained peripheral to the suite of mainstream training algorithms, and lack evaluation of analog resiliency of the trained networks.

2.4.2 Our Algorithm: HA-SGD

Intuitively, to find a flat minimum, we want the optimizer to "look around" the landscape in the neighborhood, instead of just considering the gradient at a single point.

Due to the extremely high dimensionality of the parameters in DNN models, an exhaustive inspection in the neighborhood is impracticable; instead, one can take samples in the neighborhood to collect local landscape information. Thus we propose to use Equation (2.5) to replace the conventional mini-batch gradient $\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{x}^{(i)})$ in SGD:

$$\tilde{\boldsymbol{g}}(\boldsymbol{w}) = \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)}) \tag{2.5}$$

where $\boldsymbol{x}^{(i)}$ is the i^{th} mini-batch and $\Delta\boldsymbol{\omega}^{(\ell)} \overset{iid}{\sim} q(\Delta\boldsymbol{\omega})$ are samples of deviation from the parameter point $\boldsymbol{w}$ following a distribution $q(\cdot)$.

It can be seen how the Hessian information is incorporated if we expand $\nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)})$ to the first order:

$$\nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)}) = \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{x}^{(i)}) + \mathbf{H}(\boldsymbol{w}) \Delta\boldsymbol{\omega}^{(\ell)} + \mathcal{O}\left((\Delta\boldsymbol{\omega}^{(\ell)})^2\right)$$

Thus

$$\tilde{\boldsymbol{g}}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{x}^{(i)}) + \mathbf{H}(\boldsymbol{w}) \sum_{\ell=1}^{L} \Delta\boldsymbol{\omega}^{(\ell)}$$

In the first-order, this adds a *Hessian-related* perturbation to the ordinary gradient at $\boldsymbol{w}$. Let the covariance matrix of $\Delta\boldsymbol{\omega}$ be $\boldsymbol{\Sigma}_q$. The variance of this Hessian-related perturbation is

$$\frac{1}{L} \operatorname{Tr}\left[\mathbf{H}(\boldsymbol{w}) \boldsymbol{\Sigma}_q \mathbf{H}(\boldsymbol{w})\right] \tag{2.6}$$

Thus, the optimizer will never "settle down" until it finds a local minimum where the gradient vanishes, and the Hessian-related perturbation is always low.

Based on this Hessian-related perturbation insight, we claim that the sampling number L doesn't have to be high to provide proper perturbation and enable good convergence. Thus this modification in descent step doesn't introduce much extra cost.

Also, recognizing the relation among Equations (2.2), (2.4) and (2.6), we note that it makes the Hessian-related perturbation more targeted to have the sampling q distribution similar to the analog noise distribution $p_{\Delta w}$. This implies the potential of utilizing noisy analog devices for training ADNNs.

We formalize the algorithm in Algorithm 1.

2.4.3 Results

In our HA-SGD implementation, q is chosen to be in the same form as the analog perturbation in our simulation framework. Analogous to simulation noise level, the training-noise level is defined as the ratio between standard-deviation of the perturbation and the range of the parameters per DNN component.

Effect of different training noise level

We train WideResNet-28 on CIFAR-100 with three training noise levels: 0, 0.05 and 0.1, where 0 means HA-SGD is reduced to vanilla SGD, and test them at testing noise level from

Algorithm 1: HA-SGD

Result: Perturbation-resilient parameters of the DNN, $\boldsymbol{w}^*$

Hyper-parameters: L: number of samples to take per step;
$q(\cdot)$: the perturbation distribution;
a gradient-based optimizer;
hyper-parameters of the optimizer;

initialization: $\boldsymbol{w}^{(0)}$, $i \leftarrow 0$;

while *not converge* **do**
  sample mini-batch $\boldsymbol{x}^{(i)}$ from $\mathcal{X}$;
  for $\ell \leftarrow 1$ **to** L **do**
    generate perturbation $\Delta\boldsymbol{\omega}^{(\ell)} \sim q(\Delta\boldsymbol{\omega})$;
    calculate $\nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(i)} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)})$ by forward-backward propagation;
  end
  $\tilde{\boldsymbol{g}}^{(i)} \leftarrow \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{w}} J(\boldsymbol{w}^{(i)} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)})$;
  use the optimizer and $\tilde{\boldsymbol{g}}^{(i)}$ to update $\boldsymbol{w}$: calculate $\boldsymbol{w}^{(i+1)}$;
  $i \leftarrow i + 1$;
end
return $\boldsymbol{w}^{(i)}$

0 to 0.2 with step 0.02. At each testing noise level, each trained network is instantiated 200 times by independently sampling from the device noise statistics, then each instance is tested on the test dataset on our analog-noise simulation framework, yielding a data-point. Results show that HA-SGD greatly improves the performance of the network under analog noise (Figure 2.9). At testing noise level 0.06, training noise of 0.05 increases the average top1 accuracy of the network from 53.9% to 68.0%, and training noise of 0.1 increases that to 69.2%. When the testing noise goes up to 0.12, the network trained by vanilla SGD completely fails and have top1 accuracy as random guess: 1.0%; while the network trained with 0.05 training noise level has 5.6% accuracy, and the network trained with 0.1 training noise level on average still has 43.4% accuracy.

Effect of different numbers of sampling

We further explore the effect of the hyper-parameter L: the number of gradient samples to take during HA-SGD. As shown in Figure 2.10, higher L has benefits, but only marginal. However, it has been found that larger L reduces the noise introduced and make the training

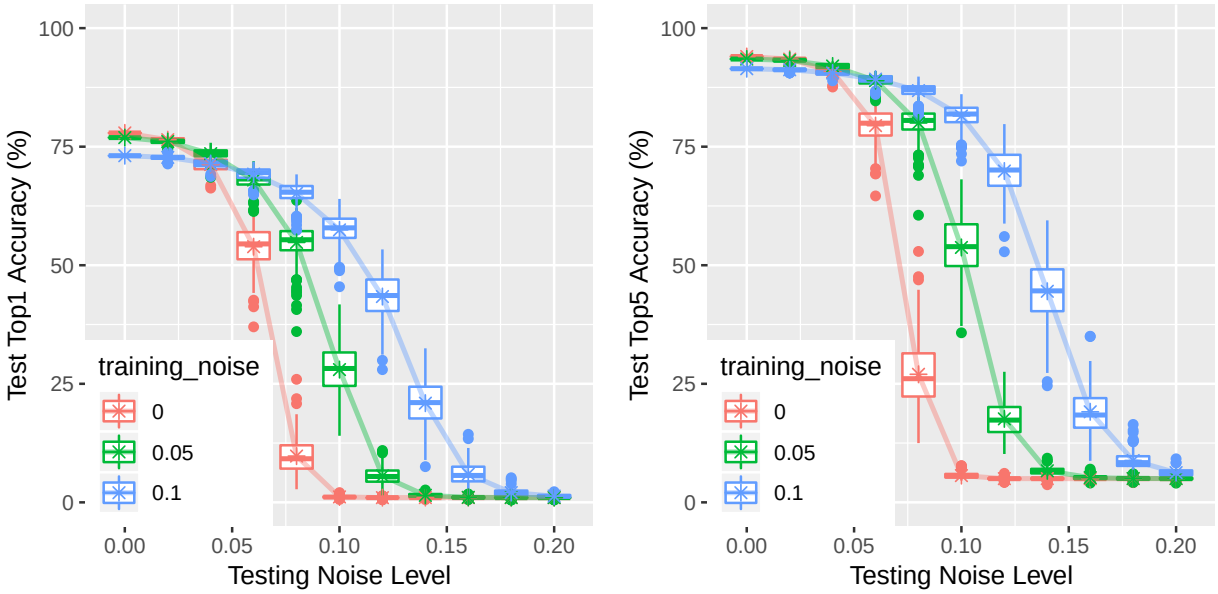


Figure 2.9: Under-noise performance comparison of networks trained with different noise (WideResNet-28 on CIFAR-100, $L = 1$). Colored lines connect the mean of test accuracy across the testing noise levels.

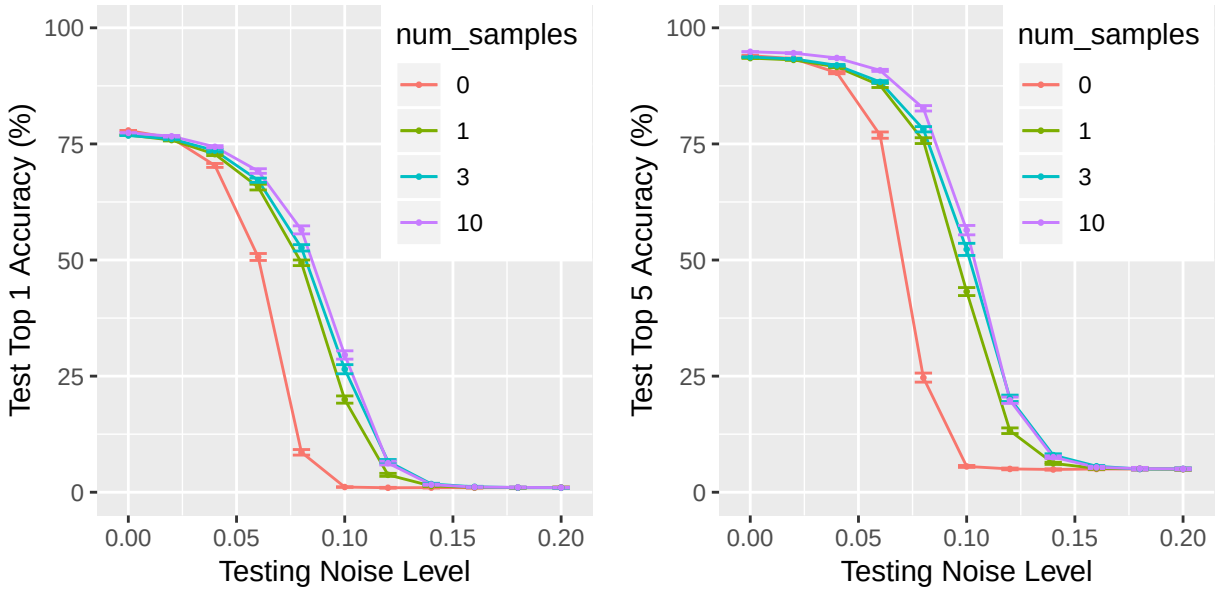


Figure 2.10: Under-noise performance comparison of networks trained with different L (WideResNet-28 on CIFAR-100, training noise level = 0.05)

easier than low L cases sometimes.

2.4.4 Smoothed landscape

In this section we provide an alternative perspective on why HA-SGD works.

Consider using $\tilde{J}(\boldsymbol{w}; \mathcal{X}) = \mathbb{E}_{q(\Delta\boldsymbol{\omega})} [J(\boldsymbol{w} + \Delta\boldsymbol{\omega}; \mathcal{X})]$ as loss function, instead of $J(\boldsymbol{w}; \mathcal{X})$. That is, instead of optimizing performance at $\boldsymbol{w}$, we optimize the expected performance at the randomly perturbed parameter $\boldsymbol{w} + \Delta\boldsymbol{\omega}$.

To optimize $\tilde{J}$, we calculate its gradient w.r.t. $\boldsymbol{w}$:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \mathcal{X}) = \nabla_{\boldsymbol{w}} \mathbb{E}_{q(\Delta\boldsymbol{\omega})} [J(\boldsymbol{w} + \Delta\boldsymbol{\omega}; \mathcal{X})]$$

Under mild assumptions for applying the Leibniz integral rule, the expectation operator and derivative operator can be swapped, giving

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \mathcal{X}) = \mathbb{E}_{q(\Delta\boldsymbol{\omega})} [\nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}; \mathcal{X})]$$

which can be estimated with L samples:

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \mathcal{X}) \approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}^{(\ell)}; \mathcal{X})$$

Its SGD version is

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \mathcal{X}) \approx \frac{1}{L} \sum_{\ell=1}^{L} \nabla_{\boldsymbol{w}} J(\boldsymbol{w} + \Delta\boldsymbol{\omega}^{(\ell)}; \boldsymbol{x}^{(i)})$$

which is exactly the step taken in HA-SGD.

From this perspective, HA-SGD is optimizing the expected performance of the DNN under random parameter-perturbation defined by q:

$$\tilde{J}(\boldsymbol{w};\mathcal{X}) = \mathbb{E}_{q(\Delta\boldsymbol{\omega})}\left[J(\boldsymbol{w}+\Delta\boldsymbol{\omega};\mathcal{X})\right] = \int J(\boldsymbol{w}+\Delta\boldsymbol{\omega};\mathcal{X})q(\Delta\boldsymbol{\omega})\,\mathrm{d}\Delta\boldsymbol{\omega} \tag{2.7}$$

This new objective is not only intuitionally meaningful, but also favors flat minima. Note that $\tilde{J}(\boldsymbol{w};\mathcal{X})$ is the original loss function $J(\boldsymbol{w};\mathcal{X})$ smoothed by a kernel defined by $q(\Delta\boldsymbol{\omega})$. After smoothing, sharp minima in the original landscape are penalize, while flat minima are still good minima (see Figure 2.11).

2.4.5 Comparison with Entropy-SGD

[CCS16] also proposed a training algorithm called "Entropy-SGD" that intentionally search for flat minima based on smoothed loss landscape:

$$\arg\min_{\boldsymbol{w}} J_{\text{ent}} = -\log \mathbb{E}_{q(\Delta\boldsymbol{\omega})}\left[\exp\left(-J(\boldsymbol{w}+\Delta\boldsymbol{\omega};\mathcal{X})\right)\right] \tag{2.8}$$

where $q(\Delta\boldsymbol{\omega})$ is chosen to be a Gaussian distribution with variance $\frac{1}{\gamma}$. The smoothing effect of J_{ent} is demonstrated in Figure 2.12.

The authors demonstrated the algorithm's ability to find flat minima by showing the sparseness of the eigen-values of the Hessian at the minima found. However, our analog resiliency evaluation will provide another good measure of the flatness of the minima found.

We trained a WideResNet-28 network on CIFAR-10 dataset using Entropy-SGD and compared its analog resiliency with the same architecture trained with our HA-SGD and vanilla SGD (see Figure 2.13). Although the differently-trained networks attain similar testing accuracy when there is no noise, they show different analog resiliency. The network trained Entropy-SGD has better resiliency than the network trained with vanilla SGD, but its resiliency is lower than the network trained with HA-SGD (training noise level is 0.1). This shows the power of HA-SGD as well as the value of our simulation framework as a minima-flatness measurement tool.

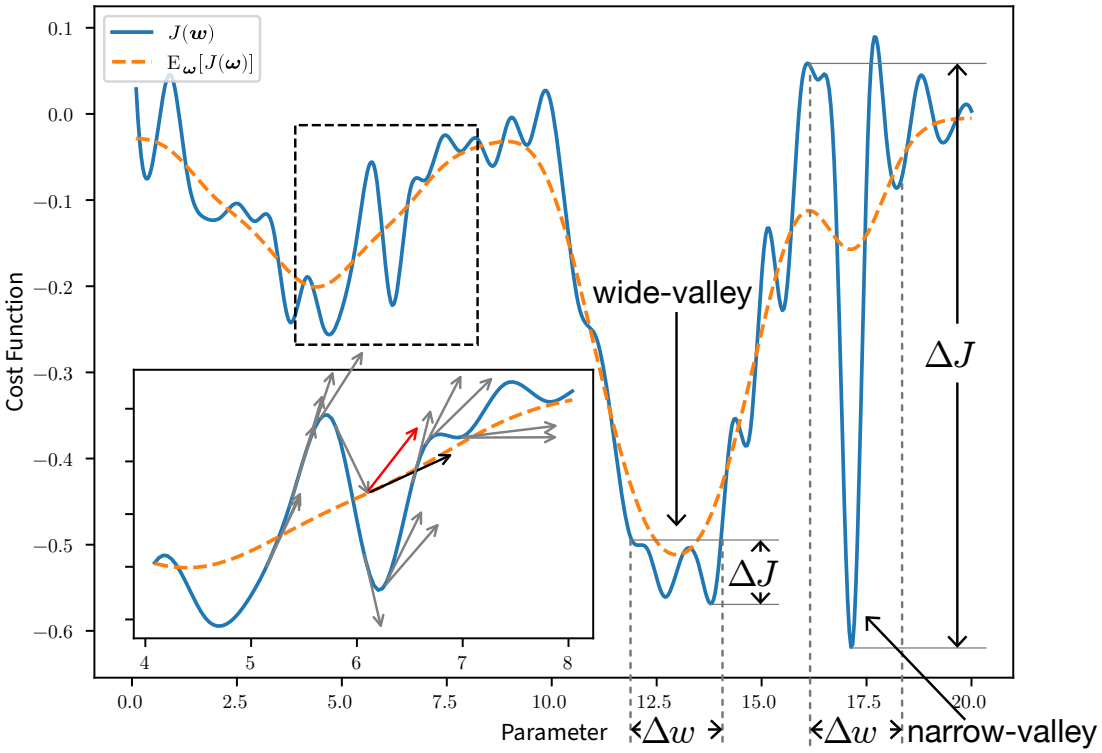


Figure 2.11: **Illustration of the smoothed-loss interpretation of HA-SGD with univariate loss function.** The solid blue curve is the original loss function; the dashed orange curve is the smoothed loss after taking expectation. **(a)**: At any $\boldsymbol{w}$, HA-SGD computes the average (the red arrow) of the sample gradients (gray arrows). This average (the black arrow) is an estimation of the gradient of the smoothed loss function $\tilde{J}(\boldsymbol{w}; \mathcal{X})$. The **slope** of each arrow shows the gradient at the point. **(b)**: The optimal training weights correspond to the local minima in the smoothed loss function. The good local minima of the smoothed loss correspond to wide-valley local minima in the original loss function.

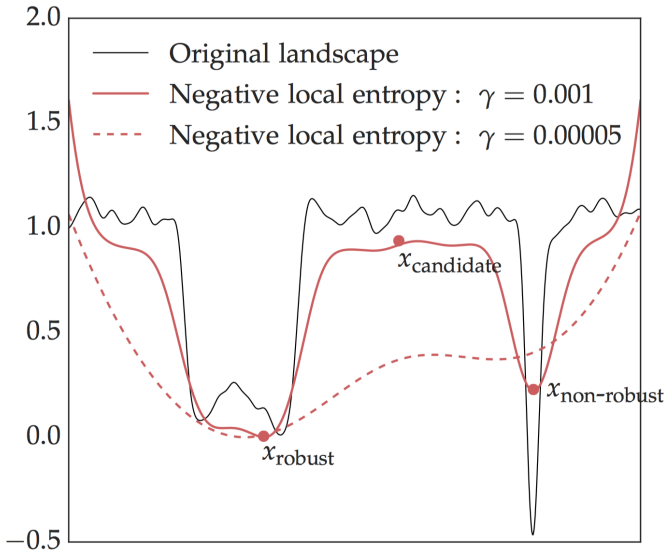
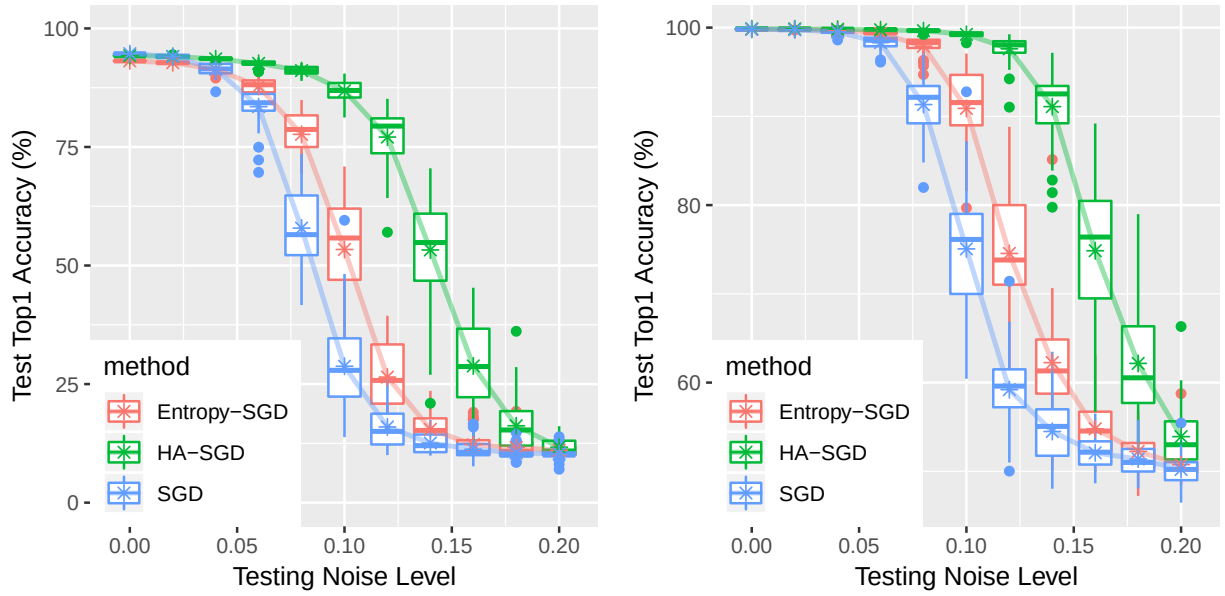


Figure 2.12: Entropy-SGD's smoothed loss function landscape illustration (taken from [CCS16])



(a) Top 1 Accuracy (b) Top 5 Accuracy

Figure 2.13: Under-noise performance comparison of networks (WideResNet-28 on CIFAR-10) trained with different method (for similar training time). Colored lines connect the mean of test accuracy across the testing noise levels.

CHAPTER 3

Physics-informed Reduced-order Modeling of Mechanical Systems — Slinky™ as a test case

3.1 Introduction

Many dynamical systems are computationally expensive to simulate with classic numerical methods based on discretized differential equations, due to the fine spatiotemporal resolution required for the discretization to hold. Reduced-order modeling (ROM) [BOC17, HM17, FMD19, MML20] provides a computationally efficient alternative to simulating complex dynamical systems with a large number of degrees of freedom. By reducing the number of variables and equations needed to describe the system, ROMs can significantly reduce the computational cost of simulation and analysis. Traditionally ROMs are built by experts with domain knowledge through repeated trial-and-error and time-consuming analysis. Alternatively, recent advances in deep learning based on deep neural (DNNs) networks [LBH15, KKL21] enable a data-driven approach to automatically construct ROMs from observed data. As demonstrated in the following, the computation time can be cut down by one to two orders of magnitude.

Slinky, a helical elastic rod, is a seemingly simple structure with unusual mechanical behavior; for example, it can walk down a flight of stairs under its weight. Taking Slinky as a test-case, we propose a physics-informed deep learning approach for building reduced-order models of physical systems. The approach introduces a Euclidean symmetric neural network (ESNN) architecture that is trained under the neural ordinary differential equation framework to learn the 2D latent dynamics from the motion trajectory of a reduced-order representation

of the 3D Slinky. The ESNN implements a physics-guided architecture that simultaneously preserves energy invariance and force equivariance on Euclidean transformations of the input, including translation, rotation, and reflection. The embedded Euclidean symmetry provides physics-guided interpretability and generalizability while preserving the full expressive power of the neural network. We demonstrate that the ESNN approach can accelerate simulation by one to two orders of magnitude compared to traditional numerical methods and achieve a superior generalization performance while classic neural networks fail to learn the Slinky dynamics, i.e., the ESNN, trained on a single demonstration case, predicts the motions accurately for unseen cases of different Slinky configurations and boundary conditions. Further investigation into the ESNN reveals that it learns the nonlinear coupling between the stretching and bending of the Slinky, which has never been captured in classical modeling.

3.2 Reduced-order Modeling of a Slinky with 2D Representation

We start by clarifying the degrees of freedom (DoFs) of a Slinky in 3D space, and subsequently, the reduced-order 2D representation that we assume.

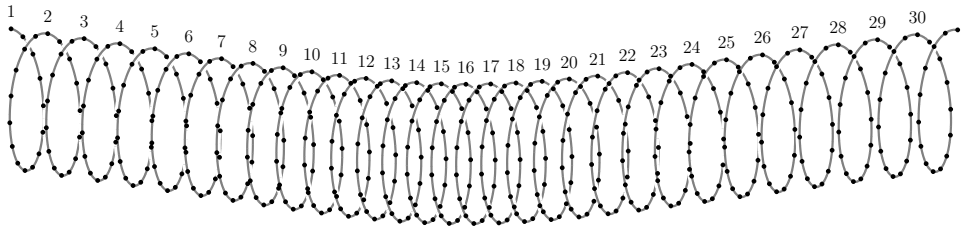


Figure 3.1: A 3D illustration of a Slinky. Numerical simulation methods discretize the helix into connected 3D nodes, resulting in a large number of DoFs. In this example, there are 16 nodes per cycle in the Slinky, each having 3 DoFs — their x, y, and z coordinates.

As shown in Figure 3.1, a Slinky is a long 3D helix consisting of many cycles. To numerically simulate the motion of a Slinky from first principles, classical methods discretize the helix into a series of nodes, approximate the stretching, curvature, and twisting along the helix, and calculate elastic energy following Kirchhoff's theory of elastic rod. Such method is known as Discrete Elastic Rod (DER) [BWR08, BAV10, KJM10, JDJ14, JNO18, PKI19].

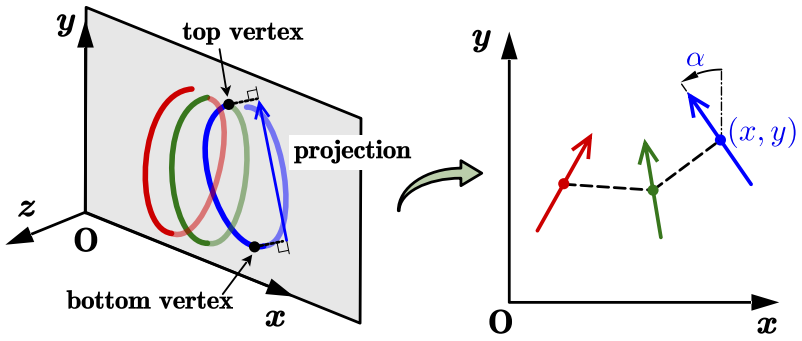
Sophisticated optimizations have been done for DER to take advantage of the band structure of the Hessian matrix of the elastic energy wrt the DoFs, allowing an efficient implicit method for simulation [JNO18].

Despite the optimizations in the DER algorithm, the large number of DoFs necessitate by the discretization introduces an inevitable computation burden, while a closer look at the motion pattern of a Slinky suggests that these DoFs from discretization could be unnecessarily many to describe the motion. In the typical motions of a Slinky, the many discretized nodes don't move individually, but collectively. Overall, a Slinky behaves like a thick elastic cylinder whose "cross-section" is a cycle in the helix, although comprising a long thin rod. This observation motivates us to describe 3D Slinkies *whose center-lines stay within a vertical plane* with 2D representations.

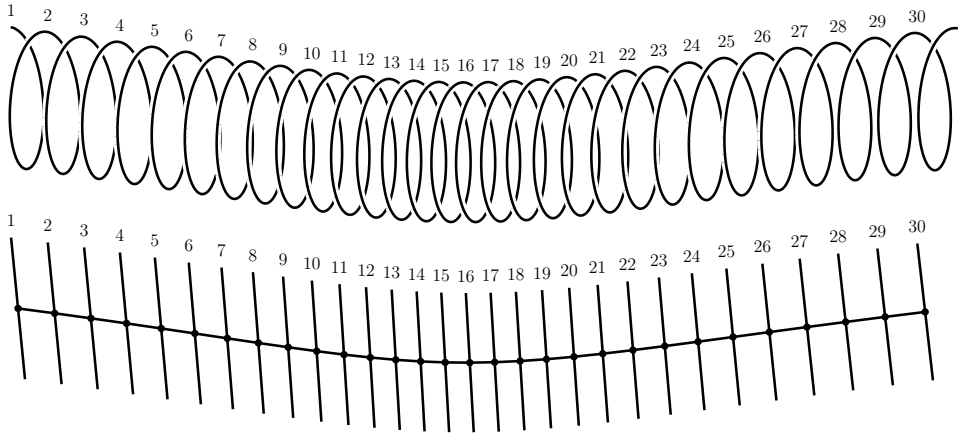
The proposed 2D Slinky representation is shown in Figure 3.2. The top and bottom vertices of a 3D Slinky cycle are projected onto the XOY plane. The vector pointing from the projected bottom vertex to the projected top vertex is the 2D representation of that cycle, referred to as a *bar* hereafter. The coordinates of the ith bar are $\boldsymbol{u}_i = \begin{bmatrix} x_i & y_i & \alpha_i \end{bmatrix}^\mathsf{T} \in \mathbb{R}^3$: x_i and y_i are the coordinates of the bar's center point, and α_i is the angle of the bar with respect to the y axis. For a Slinky of C cycles, its 2D representation has C bars, and the states of these C bars along time constitute the 2D system trajectory of the Slinky. Such a 2D reduced-order representation retains the essential geometry of a Slinky's 3D structure, based on which a 3D helical shape is still reconstructable despite the substantial reduction in the number of DoFs.

3.2.1 3D Slinky reconstruction from 2D representation

The reconstruction from 2D bar to 3D helical shape of the Slinky is shown in Figure 3.3. Given the positions of a bar, a half-circle of radius R is drawn perpendicular to the XOY plane, from the top vertex to the bottom vertex of the bar. R is the Slinky helical radius at its initial undeformed shape. Another half-circle is drawn from the bottom vertex of the bar to the top vertex of the next bar. These two half-circles form one Slinky cycle. Repeating



(a) Construction of the 2D bars by projecting 3D Slinky cycles into the XOY plane. Each cycle in the helix is simplified to a bar with 3 DoFs: the x and y of its center and the tilting angle α.



(b) An overview of the reduced-order 2D representation of a 30-cycle-long 3D Slinky.

Figure 3.2: Illustration of the reduced-order 2D representation of a 3D Slinky.

this process for all bars, a helical Slinky shape is drawn.

3.2.2 Data Schema

Dynamics modeling of Slinkies essentially involves solving the initial value problem (IVP), a time-series forecasting problem where the model's task is to predict the system's trajectory (state as a function of time) based on a given initial state. Correspondingly, we generate the time series data of Slinkies' trajectories. Each trajectory consists of the states of a Slinky at a series of time steps during its motion.

We consider a commercially-available 76-cycle Slinky (Poof-Slinky, Inc.) under large nonlinear deformation. The experiment setup is shown in Figure 3.4. We record a single Slinky

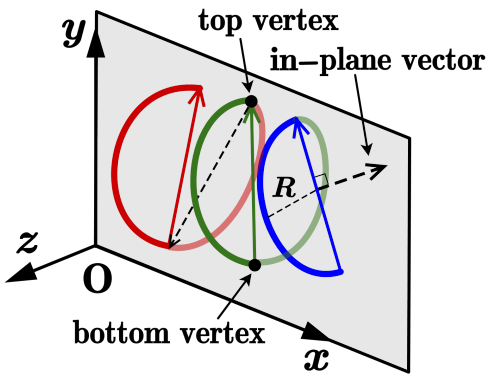


Figure 3.3: Reconstruction from 2D reduced-order model to 3D helical shape. For each bar, two half circles are drawn to form a Slinky cycle: one from the top vertex to the bottom vertex of the bar, the other from the bottom vertex of the bar to the top vertex of the next bar. The half circles are perpendicular to the XOY plane.

motion using a camera where the Slinky falls under gravity from a horizontal configuration with both ends clamped to the rack. A 3D DER model is calibrated based on the recording and used for 2D training data generation. With the calibrated model a 3D simulation is run with the same initial and boundary conditions as the experiment and with the damping removed. The 2D system trajectory is constructed based on the projection of 3D data. Aiming at data-efficient modeling, our training dataset contains only this single trajectory. Trajectories with various initial and boundary conditions are simulated for evaluation of the trained models. See supplementary materials of [LWR23] and Section 3.5 for more details.

The concrete definition of trajectories is further elaborated on in Section 3.3.1 below.

3.3 Physical Priors to Consider for Modeling a Slinky's Dynamics

Although the 2D representation preserves the essential geometry of a Slinky's 3D structure, such order reduction is not invertible, and the 2D DoFs are not physically "real". Therefore, no first principle can be directly applied to the 2D representation for dynamics modeling, making data-driven modeling a good fit. However, direct fitting with generic neural network models fails to generalize (see Section 3.6 in the following for more details). To address this problem, we incorporate several physical priors into the DNN-based modeling, which

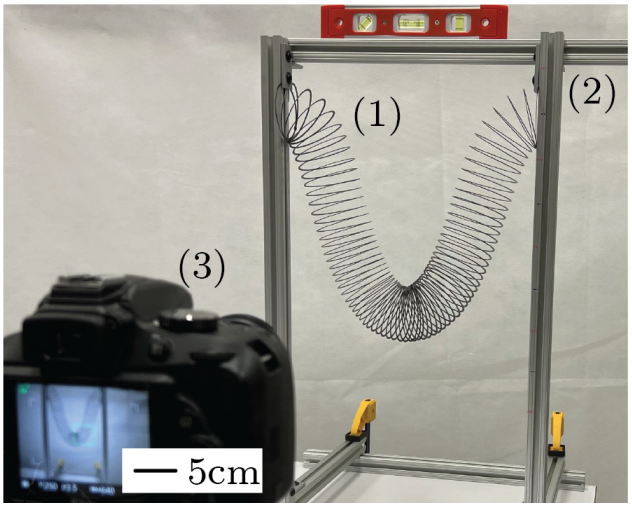


Figure 3.4: Experimental apparatus: a 76-cycle Slinky (1) is supported by an experiment frame (2). A camera (3) records the Slinky motion.

collectively amount to the modeling framework that we propose.

3.3.1 Newton's Second Law of Motion — Neural Ordinary Differential Equation

Newton's Second Law reveals that mechanical dynamical systems are second-order autonomous, described by ODEs of the form

$$\text{let } \boldsymbol{w} = \begin{bmatrix} \boldsymbol{u} \\ \dot{\boldsymbol{u}} \end{bmatrix}, \quad \dot{\boldsymbol{w}} = \begin{bmatrix} \dot{\boldsymbol{u}} \\ \ddot{\boldsymbol{u}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{u}} \\ \mathbf{M}^{-1}\,\mathbf{F}(\boldsymbol{w}) \end{bmatrix} \tag{3.1}$$

where $\boldsymbol{F}_{\boldsymbol{\varphi}}$ is the force vector containing all the internal and external forces, and $\mathbf{M}$ is the inertia matrix. Note that ODEs of any order can be rewritten into first-order ODEs by including time derivatives of the DoFs into the state vector, which enables us to express Newton's Second Law with a first-order ODE as Equation (3.1). With a set of physical parameters $\boldsymbol{\varphi}$, the force function $\boldsymbol{F}(\cdot)$ dictates the dynamics of the system, which determines a unique trajectory $\boldsymbol{w}(t)$ given an initial condition $\boldsymbol{w}(t_0)$. In the following, without loss of generality, inertia is normalized to an identity matrix, i.e., $\mathbf{M} = \mathbf{I}$. It is worth reiterating that the definition of states in ODE Equation (3.1) delineates the trajectories in the datasets considered. each trajectory is a time series of observed states, i.e. $\mathbf{T} = \begin{bmatrix} \boldsymbol{w}(t_0) & \boldsymbol{w}(t_1) & \dots & \boldsymbol{w}(t_T) \end{bmatrix}$.

Most time-series forecasting technologies from deep learning (e.g. Recurrent Neural Networks (RNNs) and Temporal Convolution Neural Networks (TCNNs)) are for generic time-series data and cannot incorporate the *a priori* knowledge of Newton's Second Law to regularize the learned model for mechanical systems. [CRB18] proposed to use a neural network $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\cdot)$ to fit data $\boldsymbol{w}(t)$ with an ODE

$$\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{w}, t) \approx \boldsymbol{f}(\boldsymbol{w}, t) = \frac{\mathrm{d}\boldsymbol{w}}{\mathrm{d}t} \iff \widehat{\boldsymbol{w}}(t) = \boldsymbol{w}(0) + \int_0^t \widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\widehat{\boldsymbol{w}}(\tau), \tau, \boldsymbol{\theta})\mathrm{d}\tau \approx \boldsymbol{w}(t) \tag{3.2}$$

Forward computation of the integral can be achieved through any suitable numerical ODE solver based on the system's requirements. When it comes to gradient-based training, one can employ the adjoint sensitivity method to calculate the gradient. This method circumvents issues common in conventional backpropagation that may arise from numerous integration steps, such as gradient vanishing or explosion and substantial memory costs. This architecture is known as Neural Ordinary Differential Equations (Neural ODEs).

Such a formulation allows Newton's Second Law to be explicitly enforced in a neural model, which enables us to introduce more physical priors to the model as explained in the following sections. A bonus from this continuous-time framework is the ability to deal with irregular observations that are not evenly spaced in time, and to predict for arbitrary time, free from the time-step resolution limitation in discrete-time frameworks.

We adopt the Neural ODE framework and learn a neural network $\boldsymbol{F}_{\boldsymbol{\theta}}$ in the place of the force function $\boldsymbol{F}_{\boldsymbol{\varphi}}$ in Equation (3.1) for the prediction of a Slinky's trajectory in terms of its 2D representation, i.e., the reduced-order modeling of its dynamics. The DoFs $\boldsymbol{u}$, in this case, is the 2D representation of the Slinky's configuration, i.e. $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u}_1^\mathsf{T} & \dots & \boldsymbol{u}_C^\mathsf{T} \end{bmatrix}^\mathsf{T}$, and physical parameters $\boldsymbol{\varphi}$ here include the geometry, the elasticity and the density of the Slinky, along with gravitational acceleration and the damping factor.

3.3.2 The Locality of Elastic Energy and the Homogeneity of Slinkies

The dynamics of a Slinky entailed by the neural network involves elastic force, gravity, collision, and damping, among which the elastic force and gravity play the major roles. Gravity requires no learning, and consequently the major task of the data-driven neural model is to learn the elastic force $\boldsymbol{F}^E$, which is attributed to local deformation, and therefore only requires local geometry information to predict.

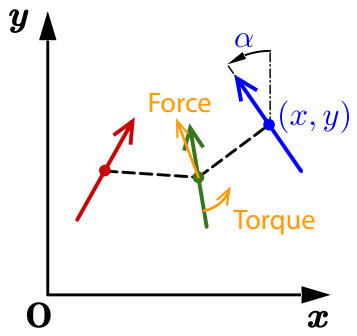
We hypothesize that the elastic energy of one cycle of the Slinky is essentially determined by the three-cycle segment of the cycle and its two adjacent cycles. Correspondingly in the 2D representation, we expect to be able to predict the elastic force experienced by the i^{th} *bar* using the configuration of only three bars, $\begin{bmatrix}\boldsymbol{u}_{i-1}^\mathsf{T} & \boldsymbol{u}_i^\mathsf{T} & \boldsymbol{u}_{i+1}^\mathsf{T}\end{bmatrix}$. Hereafter, such a group of three adjacent bars is referred to as a *triplet*. At the ends of the Slinky, "ghost bars" are prepended or appended to allow such triplet-based force prediction for the end bars (see Section 3.4.3 for more details).

A Slinky is a uniform long helix (apart from its two ends), therefore the cycles should be identical and share the same dynamics, which means the same neural elastic force predictor should apply to all cycles.

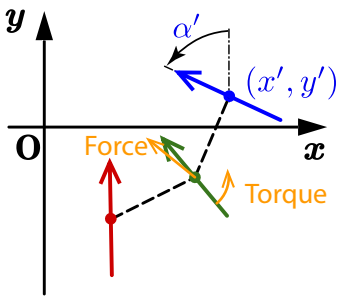
3.3.3 Euclidean Symmetry

Because elastic forces are entirely determined by the geometrical configuration of a Slinky, they should remain invariant in a local reference frame of the Slinky under *rigid body rotation* and translation. In other words, when the Slinky is rotated with its geometrical configuration unchanged, in the global reference frame, the internal elastic forces should also rotate in the same way. Also, suppose the Slinky's configuration is mirrored, the elastic forces should be mirrored accordingly. Formally, in the case of 2D Slinky representations, given any element from the 2D rotation group, i.e. a rotation matrix $\mathbf{R} \in \mathrm{O}(2)$,

$$\boldsymbol{F}^E(\mathbf{R}\boldsymbol{u}) = \mathbf{R}\boldsymbol{F}^E(\boldsymbol{u}), \tag{3.3}$$



(a) Configuration of three bars in a Slinky's 2D representation based on which the force and torque experienced by the middle bar can be computed.



(b) Rotated and translated configuration of the three bars. The force and torque should transform accordingly

Figure 3.5: Demonstration of Euclidean equivariance: rotation equivariance and translation invariance.

where $\mathbf{R}\boldsymbol{u}$ means the Slinky DoFs $\boldsymbol{u}$ rotated by the group element $\mathbf{R}$. Similarly, the elastic forces should be invariant under arbitrary translation applied to the entire Slinky. Considering three bars as described in Section 3.3.2, these properties are demonstrated in Figure 3.5.

The group of all possible combinations of translation and rotation in n-dimensional space form the so-called Euclidean group $\mathrm{E}(n)$, thus the rotation equivariance and translation invariance sometimes are collectively referred to as *Euclidean equivariance* [SHW21, GS22].

Euclidean equivariance is important prior knowledge because it allows a model to generalize to unseen configurations by exploiting symmetry, which proves to be vital to the data efficiency of a deep learning model in ablation studies (Section 3.6).

However, equivariance is not easy to enforce. Novel neural network architectures that guarantee various equivariance are still a popular research topic [TSK18, CWK19, FWW21, HHR21, SHW21, GS22], and current general-purpose equivariant neural network models inevitably sacrifice model's flexibility or performance for equivariance. In this work, considering the conservative nature of elastic force, we induce force equivariance from the invariance of elastic energy. Concretely, $\forall \mathbf{R} \in \mathrm{O}(2)$, if the elastic energy $E(\boldsymbol{u})$ is invariant under the transformation of $\mathbf{R}$,

$$E(\mathbf{R}\boldsymbol{u}) = E(\boldsymbol{u}), \tag{3.4}$$

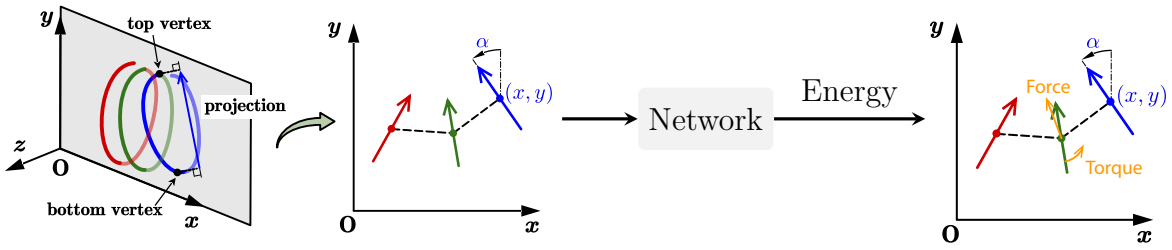


Figure 3.6: The neural network module takes three bars of the 2D representation as input, and gives the force and torque for the middle bar by automatic differentiation of the surrogate energy.

then the elastic force $\boldsymbol{F}^E(\boldsymbol{u}) = -\nabla_{\boldsymbol{u}} E(\boldsymbol{u})$ is naturally equivariant,

$$-\nabla_{\boldsymbol{u}} E(\mathbf{R}\boldsymbol{u}) = -\nabla_{\boldsymbol{u}} E(\boldsymbol{u}) \tag{3.5}$$

$$-\mathbf{R}^{\mathsf{T}} \left(\nabla_{\boldsymbol{u}} E\right) (\mathbf{R}\boldsymbol{u}) = -\nabla_{\boldsymbol{u}} E(\boldsymbol{u}) \tag{3.6}$$

because $\mathbf{R}\mathbf{R}^{\mathsf{T}} = \mathbf{I}$,

$$-\left(\nabla_{\boldsymbol{u}} E\right) (\mathbf{R}\boldsymbol{u}) = -\mathbf{R}\nabla_{\boldsymbol{u}} E(\boldsymbol{u}) \tag{3.7}$$

$$\boldsymbol{F}^E(\mathbf{R}\boldsymbol{u}) = \mathbf{R}\boldsymbol{F}^E(\boldsymbol{u}) \tag{3.8}$$

Note that the $\mathbf{R}$ could involve the transformation of a mirror reflection in addition to a rotation. The translation invariance is also guaranteed in this way as any translation — a constant offset in $\boldsymbol{u}$ — vanishes when taking the gradient ($\nabla_{\boldsymbol{u}}$).

In practice, we train a neural network to take a Slinky's DoFs as input and produce a scalar "surrogate energy" as output, from which the corresponding elastic force is calculated as the derivative of this scalar energy with respect to the input DoFs, using PyTorch's automatic differentiation. Overall, the input and output of the neural network is illustrated in Figure 3.6.

Local Coordinates for Energy Invariance

To achieve Euclidean invariance of the neural network output, a set of relative coordinates of both rotation and translation invariance are adopted.

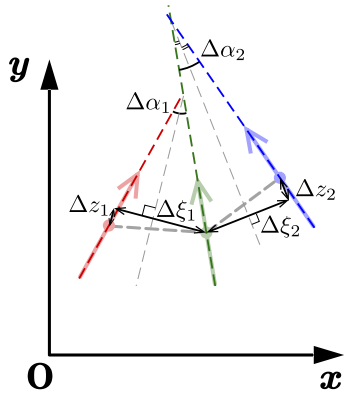


Figure 3.7: Transformation of global coordinates of a triplet into relative coordinates. ζ_{i-1} is parallel to the angular bisection between α_i and α_{i-1}.

Consider a single input of the network, i.e., the global coordinates of the ith triplet

$$\boldsymbol{\mu}_i = \begin{bmatrix} \boldsymbol{u}_{i-1}^\mathsf{T} & \boldsymbol{u}_i^\mathsf{T} & \boldsymbol{u}_{i+1}^\mathsf{T} \end{bmatrix} = \begin{bmatrix} x_{i-1} & y_{i-1} & \alpha_{i-1} & x_i & y_i & \alpha_i & x_{i+1} & y_{i+1} & \alpha_{i+1} \end{bmatrix}^\mathsf{T} \in \mathbb{R}^9. \quad (3.9)$$

For simplicity, we omit the triplet index i and denote $\boldsymbol{\mu}_i$ as $\boldsymbol{\mu}$. To make the representation invariant to rigid body translation and rotation, the global coordinates $\boldsymbol{\mu}$ are transformed into relative coordinates $\boldsymbol{z} = \begin{bmatrix} \Delta\xi_{i-1} & \Delta\zeta_{i-1} & \Delta\alpha_{i-1} & \Delta\xi_i & \Delta\zeta_i & \Delta\alpha_i \end{bmatrix} \in \mathbb{R}^6$, where

$$\Delta\xi_i = \cos\left(\frac{\alpha_{i+1}+\alpha_i}{2}\right)(x_{i+1}-x_i) + \sin\left(\frac{\alpha_{i+1}+\alpha_i}{2}\right)(y_{i+1}-y_i) \quad (3.10)$$

$$\Delta\zeta_i = -\sin\left(\frac{\alpha_{i+1}+\alpha_i}{2}\right)(x_{i+1}-x_i) + \cos\left(\frac{\alpha_{i+1}+\alpha_i}{2}\right)(y_{i+1}-y_i) \quad (3.11)$$

$$\Delta\alpha_i = \alpha_{i+1} - \alpha_i \quad (3.12)$$

See Figure 3.7 for an illustration of relationships among the quantities.

What remains to cover in Euclidean symmetry is the reflection invariance. To achieve that, $\mathbf{z}$ and its 3 reflected copies are passed through the neural network $f_{\boldsymbol{\theta}}$. The outputs are 4 scalars, and their summation is the energy surrogate E:

$$E(\mathbf{z}) = f_{\boldsymbol{\theta}}(\mathbf{z}) + f_{\boldsymbol{\theta}}\left(\mathsf{Refl}_x(\mathbf{z})\right) + f_{\boldsymbol{\theta}}\left(\mathsf{Refl}_y(\mathbf{z})\right) + f_{\boldsymbol{\theta}}\left(\mathsf{Refl}_x(\mathsf{Refl}_y(\mathbf{z}))\right) \quad (3.13)$$

where $\mathsf{Refl}_x(\cdot)$ and $\mathsf{Refl}_y(\cdot)$ stand for the reflection of the triplet coordinates about x and y axes, respectively. Note that the reflections are self-inverse and commutative, i.e.,

$$\mathsf{Refl}_x(\mathsf{Refl}_x(\cdot)) = I(\cdot),\ \mathsf{Refl}_y(\mathsf{Refl}_y(\cdot)) = I(\cdot),\ \text{and}\ \mathsf{Refl}_x(\mathsf{Refl}_y(\cdot)) = \mathsf{Refl}_y(\mathsf{Refl}_x(\cdot)) \tag{3.14}$$

where $I(\cdot)$ is the identity transformation. Therefore, the energy surrogate E satisfies the following property:

$$E(\mathbf{z}) = E(R_x(\mathbf{z})) = E(R_y(\mathbf{z})) = E(R_x(R_y(\mathbf{z}))) \tag{3.15}$$

In summary, by feeding the neural network reflected copies of *relative coordinates*, the Euclidean invariance of the scalar output surrogate energy is guaranteed.

3.3.4 Conservative Force and Energy Conservation

Another benefit in computing the force prediction as derivatives of a scalar is that such force is guaranteed to be a conservative force that corresponds to a potential energy (a scalar field in the configuration space). Motions driven by such force following Equation (3.1) have conserved mechanical energy (the sum of potential energy and kinetic energy is a constant). Therefore, the dynamics prescribed by the neural can be regularized and divergence can be avoided by regularizing the surrogate potential energy.

The aforementioned approach using the derivatives of a scalar neural function to predict vector quantities can be seen as a special case of Lagrangian Neural Networks [LRP18, CGH20], which also aims to introduce the energy conservation constraints to neural networks for modeling dynamical systems.

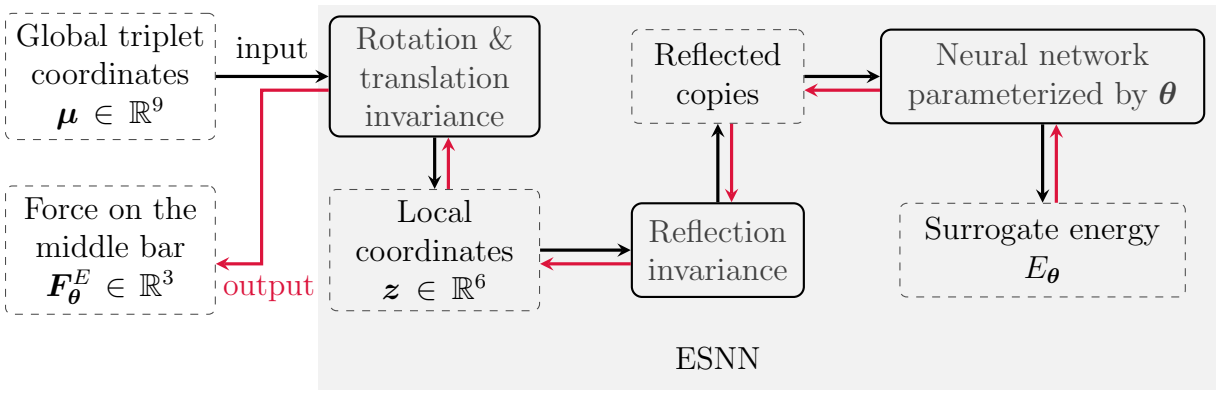


Figure 3.8: The workflow of Euclidean symmetric neural network (ESNN). Dashed frames enclose quantities, and solid boxes denote processes transforming the quantities (and introducing symmetries). Black arrows are forward computation towards the surrogate energy, and red arrows are corresponding backpropagation towards the force output.

3.4 Modeling Framework: Euclidean Symmetric Neural ODE

3.4.1 Neural Network Architecture

To sum up, the neural network force predictor that enjoys Euclidean equivariance by incorporating the aforementioned techniques is illustrated in Figure 3.8.

The neural network used is a multilayer perceptron (MLP) with skip connections inspired by DenseNet [HLv16]. For activation function, Softplus proves to have better performance compared to ReLU. To regularize the potential energy landscape, the final output uses a square activation.

3.4.2 The DenseNet-like Backbone

The backbone in the Euclidean symmetric neural network (ESNN) is a DenseNet-like structure, where shortcut pathways are created for a layer from all its previous layers. The backbone network receives a $\boldsymbol{z} \in \mathbb{R}^6$ vector as input. It first increases the feature dimension to 32 by a fully-connected (FC) layer. Then the feature is passed through FC layers with Softplus activation. The new feature with an increased dimension is formed by concatenating

the previous feature with the FC layer output, i.e.,

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{y} \\ \mathbf{x}_{i-1} \end{bmatrix} = \begin{bmatrix} \mathrm{FC}_{\mathrm{Softplus}}(\mathbf{x}_{i-1}) \\ \mathbf{x}_{i-1} \end{bmatrix} \tag{3.16}$$

After 5 layers, the feature dimension is increased to 192. This feature is then squared and passed through an FC layer with no activation to produce the final scalar output.

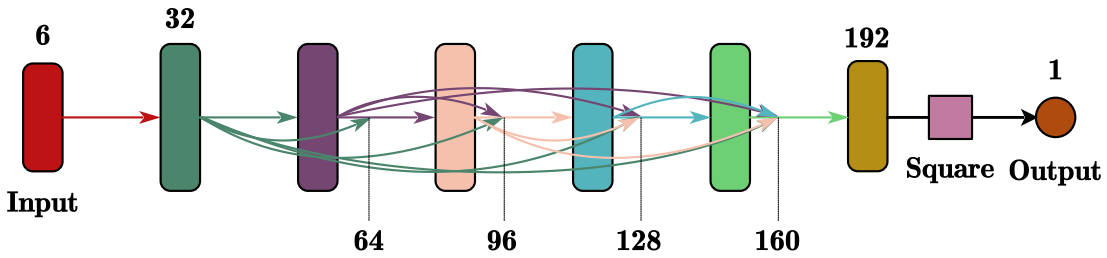


Figure 3.9: The DenseNet-like network first increases the feature dimension from 6 to 32 by an FC layer without activation, and then in the subsequent 5 layers, concatenating the feature in a layer with the output of an FC layer to produce the feature for the next layer. This is equivalent to creating shortcuts for a layer from all of its previous layers. The feature dimension will increase by 32 after passing through each layer and eventually reach 192. Then the feature is squared and passed through an FC layer without activation to generate a scalar output.

3.4.3 Bounrday Condition Treatment

When elastic force prediction is needed for the boundary bars at the ends of a Slinky, extra "ghost bars" are added to the ends to form triplets of which the boundar bars are the "middle bar".

For clamped boundary conditions, no special treatment is needed since it is unnecessary to calculate the elastic force on the boundary bar. For free boundary conditions, a ghost bar is constructed, as shown in Fig. 3.10. Assume that the N^{th} bar of a N-cycle Slinky is free. Then the coordinates of the ghost bar (the $(N+1)^{\mathrm{th}}$ bar) are the same as those of the N^{th} bar, i.e., $(x_{N+1}, y_{N+1}, \alpha_{N+1}) = (x_N, y_N, \alpha_N)$. The elastic force on the Nth boundary bar is

calculated with

$$(x_{N-1}, y_{N-1}, \alpha_{N-1}, x_N, y_N, \alpha_N, x_{N+1}, y_{N+1}, \alpha_{N+1})$$

as the ESNN input.

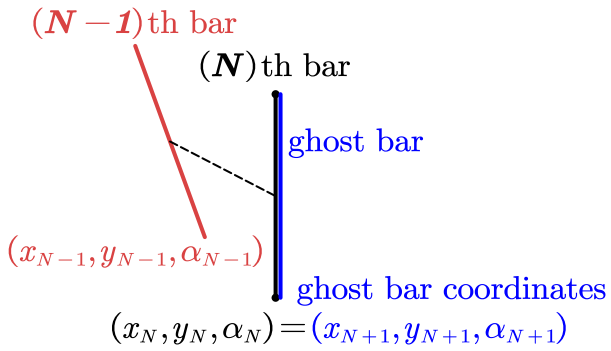
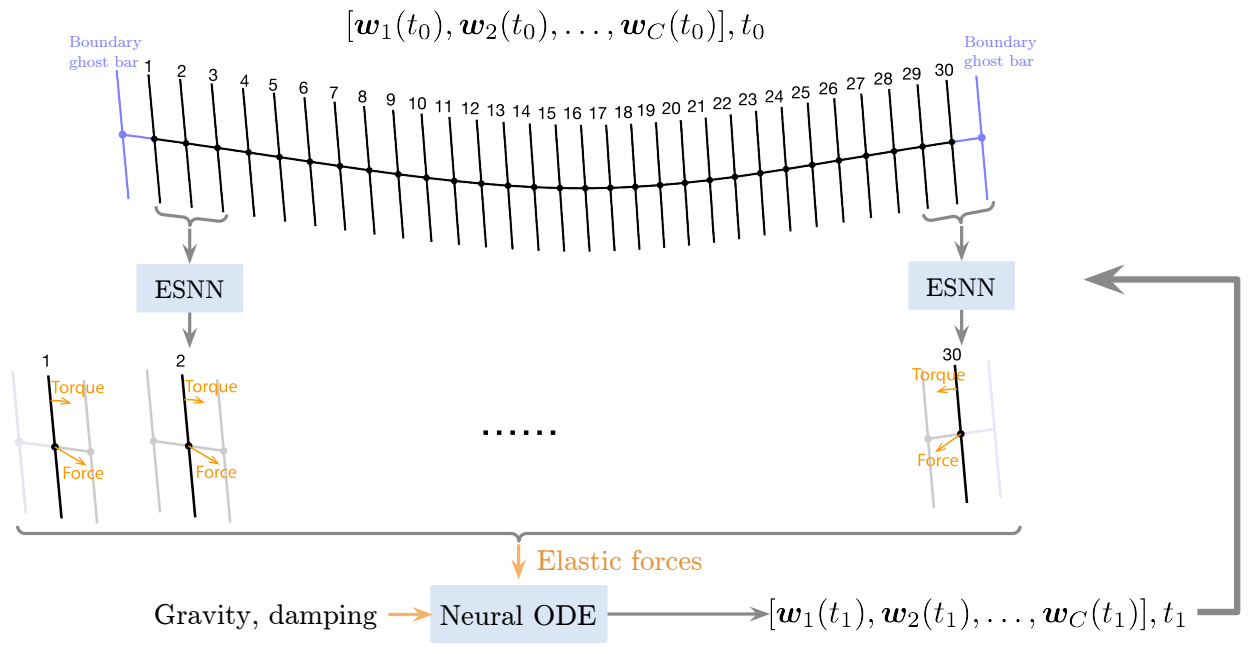


Figure 3.10: The boundary treatment for free-end boundary conditions. The ghost bar (the $(N+1)^{\text{th}}$ bar) has the same coordinates as the N^{th} bar.

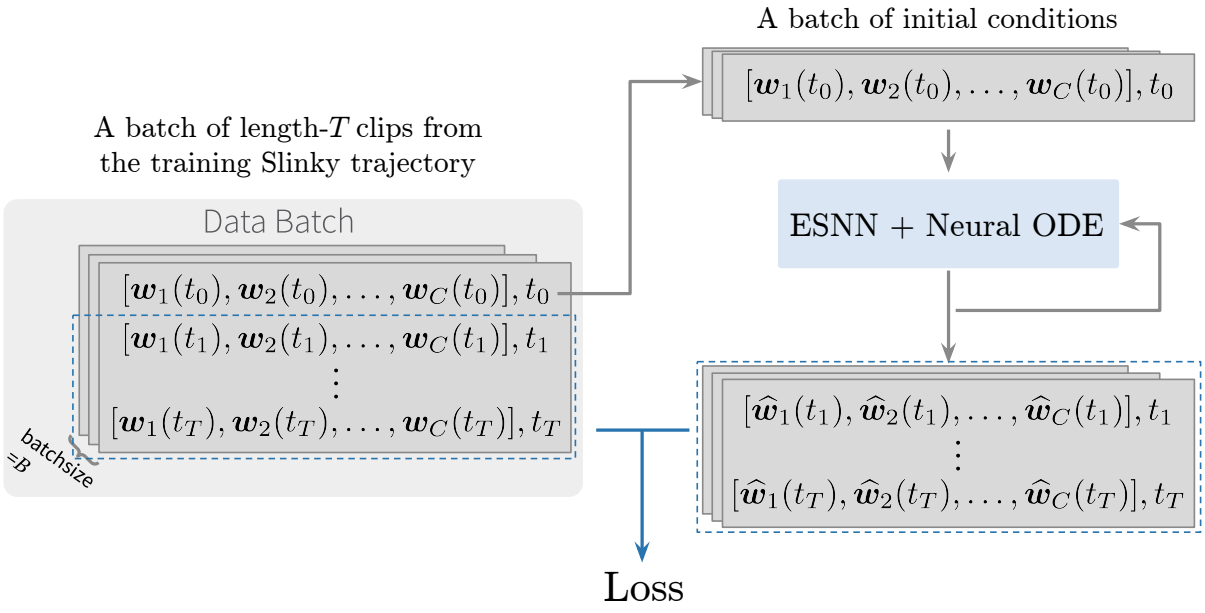
3.4.4 Training Workflow

Figure 3.11 illustrates the overall training workflow of ESNN with Neural ODE. Given a batch of clips in the training trajectory, comprising T steps of states, the predictions for later time steps are autoregressively generated starting from the initial time step with ESNN and Neural ODE. The loss is calculated by comparing the predicted trajectory with the groundtruth. The ESNN model is trained to optimize such loss with Adam optimizer [KB14].

During ESNN training, the trajectory clip length T is gradually increased from 2 to 70 to let the model gradually learn long-term effect of its instantaneous predictions while keeping the loss stable. As described in Section 3.2.2 The simulation time step is 0.01 s, making the final training trajectory time span 0.7 s. A damping and a contact model [LFS20] are added in the ESNN deployment to match the real-world energy dissipation and non-penetration. The ESNN is trained on this one case, and tested on other unseen cases.



(a) Illustration of state update from time step t_0 to t_1. The predicted force changes along the state, which in turn influences the state's evolution. Such a continuous process is solved by integrating the Neural ODE prescribed by the ESNN.



(b) The training diagram of ESNN with Neural ODE.

Figure 3.11: The training schematic of the dynamics modeling framework comprising ESNN and Neural ODE.

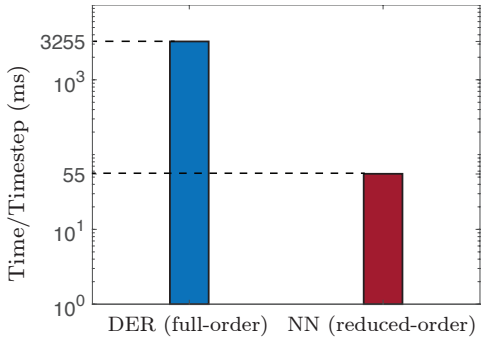


Figure 3.12: Computation time comparison between 3D DER simulation and ESNN reduced-order model.

3.5 Generalizable Dynamics Learned from Single Trajectory

3.5.1 Trajectory Prediction Beyond Training Time Span

After 1000 training epochs, a 2D simulation is run with the trained ESNN for 2.5 s to evaluate its ability to predict trajectories for the same Slinky. Recall that the training data contains only the first 0.7 s of the trajectory. Therefore we refer to the first 0.7 s span of the predicted trajectory the "train phase" and the 0.7 s to 2.5 s span the "extrapolation phase". The predicted trajectory of the 2D Slinky is compared with 3D DER simulation visually in Figure 3.13(b). The ESNN results match the ground truth well not only within the training phase, but also in the extrapolation phase up to 2.5 s, which is more than 2 times the training time span. Note that extrapolation here refers to time-domain forecasting rather than making predictions on "out-of-distribution" data. The final static deformations for real-world experiment, DER, and ESNN are in good agreement, as shown in Figure 3.13(a). In terms of computational speed, the ESNN outperforms the DER method by roughly 60 times as shown in Figure 3.12.

3.5.2 Generalizable Prediction in Various Unseen Cases

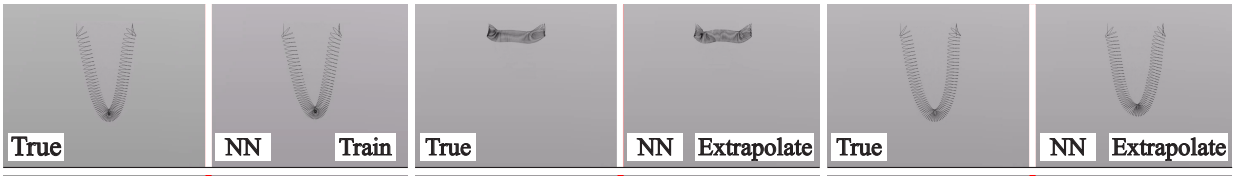
With the weights and all physical parameters fixed, we test the generalization of the ESNN on previously unseen cases: on a Slinky (1) of a different number of cycles; (2) under a different boundary condition; (3) of a different density; and (4) of a different

Young's modulus. (refer to Supplementary Materials for the videos of the comparison). In test case (1), the Slinky cycle number is changed to 40 with both ends clamped, as shown in Figure 3.13(c). The Slinky is initially held horizontally and then dropped freely under gravity. The motion time shots of the Slinky from the real-world experiment, DER simulation, and ESNN simulation are compared in the Figure 3.13(c) and show an excellent agreement. In test case (2), the 40-cycle Slinky is held vertically, clamped at the upper end, and dropped freely from its undeformed configuration. The motion time shots from different methods are compared in the Figure 3.13(d) and again we observe a good agreement. In these two test cases, a satisfactory agreement is achieved without making any changes to the ESNN. The agreement originates from the embedded Euclidean symmetry and the fact that the NN is learning generalizable physics locally. In contrast, [LKA20] and [HRM21] construct NN-based surrogate models for full-field solutions under a certain boundary condition. The benefit is that the NN prediction is extremely fast since only one forward pass through the trained NN is required for each prediction. The price paid is in the generalization ability. For a new type of boundary condition, a new dedicated training dataset is required and the NN needs to learn from scratch. However, in our ESNN approach, different boundary conditions can be readily incorporated after training since the ESNN learns local physics which is boundary condition agnostic. In test case (2), the orientation of the Slinky is shifted by 90 degrees. The ESNN is still capable of generating the correct surrogate forces and system trajectory prediction since it preserves rotation equivariance.

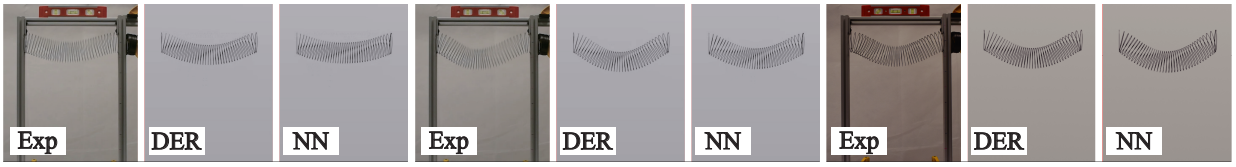
3.6 Ablation Study

In this section, we compare the ESNN Neural ODE approach to various data-driven dynamics modeling baselines to understand the effectiveness and necessity of the prior knowledge and principles incorporated in the method, which enables highly data-efficient training with a *single demonstration* and reliable generalize to unseen scenarios where *both initial and boundary conditions are varied*.

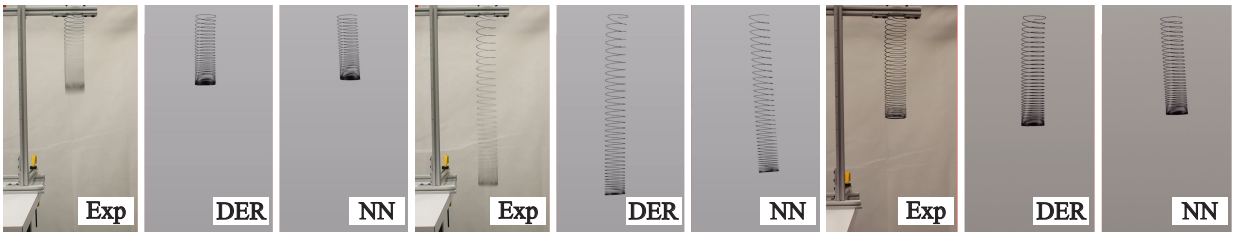
(a) Static deformation comparison across a real-world Slinky experiment, the discrete elastic rod (DER) simulation, and the ESNN reduced-order model.



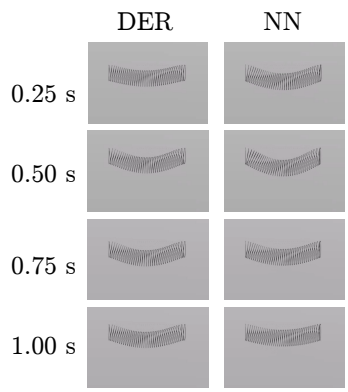
(b) Comparison between 3D DER ground truth and reconstruction from the ESNN reduced-order model in the training phase and extrapolation phase. Time of shots are at 0.45 s, 2.03 s, and 2.50 s.



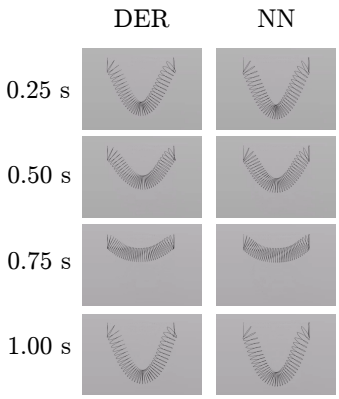
(c) Comparison of an unseen 40-cycle Slinky across real-world experiment, 3D DER simulation, and reconstruction from the ESNN reduced-order model. Time of shots are at 0.53 s, 1.43 s, and static.



(d) Comparison of testing on a 40-cycle Slinky with a different boundary condition and orientation. Time shots are at 0.2 s, 0.53 s, and static.



(e) Comparison of testing on a 40-cycle Slinky with half of the material density.



(f) Comparison of testing on a 40-cycle Slinky with half of the Young's modulus.

Figure 3.13: Comparison across real-world experiments, 3D DER simulations and ESNN 2D simulations.

3.6.1 The Importance of Physical Knowledge

To provide a physics-agnostic, data-driven baseline for comparison, a convolutional neural network (CNN) architecture is used to directly learn the one-step dynamics of the Slinky. Given the 2D coordinates of a Slinky at a time step, the CNN predicts the 2D Slinky coordinates at the next time step using a series of non-linear filters convoluting along the Slinky cycles with a receptive field of three cycles. Such a convolutional architecture assumes homogeneity of the Slinky across cycles (as other methods do) and allows the application of the trained model to Slinkies consisting of an arbitrary number of cycles. Boundary condition treatment is the same as that used in the ESNN. As shown in Figure 3.14(b), the training result is fairly accurate. After training, the CNN is tested on a 40-cycle Slinky vertically held at the upper boundary. The unreasonable result in Figure 3.14(c) shows that the CNN "memorizes" the data pattern in the training dataset, but fails to learn the underlying dynamics, therefore does not generalize to unseen conditions (Figure 3.14(f)).

Two other comparison methods are the "direct force predictor" and the "direct energy predictor". The schematics of the methods are shown in Figure 3.14(a). The NNs in the direct force predictor method (referred to as "force NN") and in the direct energy predictor method (referred to as "energy NN") share the same structure as the ESNN in terms of the number of layers and neurons per layer. See Section 3.6.2 for training data preparation for these two methods.

3.6.2 Data generation for "direct force predictor" and "direct energy predictor" methods

After the 3D simulation is run, the total energy of each discrete elastic rod at each time step, including bending, stretching, and twisting energy, is recorded. The total energy of each Slinky cycle is the summation of the energies from the associated discrete rods. The 2D triplet coordinates and the corresponding total energy of the middle cycles constitute the inputs and outputs of the training data pairs for the direct energy predictor method. The

NN in the direct energy predictor method is supposed to learn the mapping between triplet coordinates and middle bar energies.

After converting the 3D Slinky trajectory to 2D bar trajectory, the instantaneous acceleration of each bar is calculated with the finite difference method. The pseudo elastic force on each bar is calculated by Newton's second law of motion. The NN in the direct force predictor method is supposed to directly learn the mapping between triplet coordinates and middle bar elastic forces.

3.6.3 The Importance of Euclidean Symmetry

The force NN is trained to directly learn the mapping between the coordinates of a triplet and the elastic forces on its middle bar without equivariance enforced in the NN architecture. Although a reasonably low loss is achieved for training, the simulation that solves the motion equation Equation (3.1) with the force NN as the elastic force function fails to converge (Figure 3.14(d)). Without the equivariance property, the force NN fails to generate physically reasonable and consistent predictions on unseen inputs. A natural thought to solve this issue is to train a more capable NN on an augmented dataset with rotation and chiral transformations. This approach is unrealistic because the rotation angle is a continuous variable, making the amount of data augmentation required to enforce rotation-equivariance prohibitively huge. This highlights the necessity of incorporating physical symmetries to improve the efficacy and efficiency of NN-based reduced-order models.

3.6.4 The importance of the NODE training framework

The results from the energy NN are shown in Figure 3.14(e). The energy NN is trained to predict the elastic energy given the coordinates of a triplet. Similar to the ESNN, the elastic force on the middle bar, which is the negative energy gradient w.r.t. the coordinates of the middle bar, can be accordingly derived with backpropagation. The energy NN differs from the ESNN in the training scheme: the energy NN is trained on coordinates-energy data pairs; the backpropagation to calculate forces is performed during the testing phase for

motion simulation. Recall that for the ESNN, the backpropagation is already embedded in the training phase under the NODE framework to fit the Slinky trajectory. The energy NN simulation diverges at 10.7 ms. The divergence is due to two reasons: (1) Although the energy NN achieves a nearly perfect energy fitting, there is no guarantee that the derivatives of the predicted energy, i.e. predicted forces, are fitted correctly, especially in a high dimensional space, with limited training data; (2) There will inevitably be errors in the energy prediction of the NN and the associated force calculation after training. The errors after propagating through an ODE solver should not diverge the simulation. This property is guaranteed when the NN is trained under the NODE framework, but not when directly fitting input-output data pairs. This highlights the necessity of using NODE as the training scheme for NN-based reduced-order models.

3.7 Comparison with classic 2D Slinky models

By investigating the prediction of the intermediate energy surrogate of the ESNN, we show that it automatically discovers the nonlinear stretching-bending coupling of the Slinky without any prior knowledge. Figure 3.14(g) shows the triplet energy at different bending angles with 3 initial stretching lengths. As the initial stretching length increases, it takes more energy to bend the bars to a certain angle, i.e., the instantaneous bending stiffness is a function of the stretching length, which is a physical insight not reflected in classic models [HBM14].

3.8 Conclusion

In this study, we developed a highly generalizable and data-efficient, deep-learning-based reduced-order dynamics model for Slinkies. This method leads to a 60-fold acceleration in Slinky motion simulations compared to finite-element numerical methods. The model successfully learns reduced-order physics from a single demonstration case and delivers numerically stable, accurate, and rapid predictions on a range of unseen cases. The cornerstone

of the method is the ESNN, trained with the Neural ODE framework, which enforces physical principles, specifically Euclidean symmetry and Newton's second law of motion. A series of ablation studies comparing the ESNN with alternative data-driven deep learning methods underscored the effectiveness and importance of our approach's components. Furthermore, we demonstrated that the ESNN automatically identifies the phenomenon of nonlinear stretching-bending coupling that manifests in a Slinky's dynamics, which has never been accounted for in classical reduced-order studies.

3.9 Additional Information

This research was collaboratively conducted with Dr. Qiaofeng Li during his post-doctoral appointment in the Structures-Computer Interaction group at UCLA led by Prof. Khalid Jawed. The findings have been published in Extreme Mechanics Letters, notably featured on the journal's cover [LWR23]. The original code repository of this work is released on Github at https://github.com/StructuresComp/slinky-is-sliding.

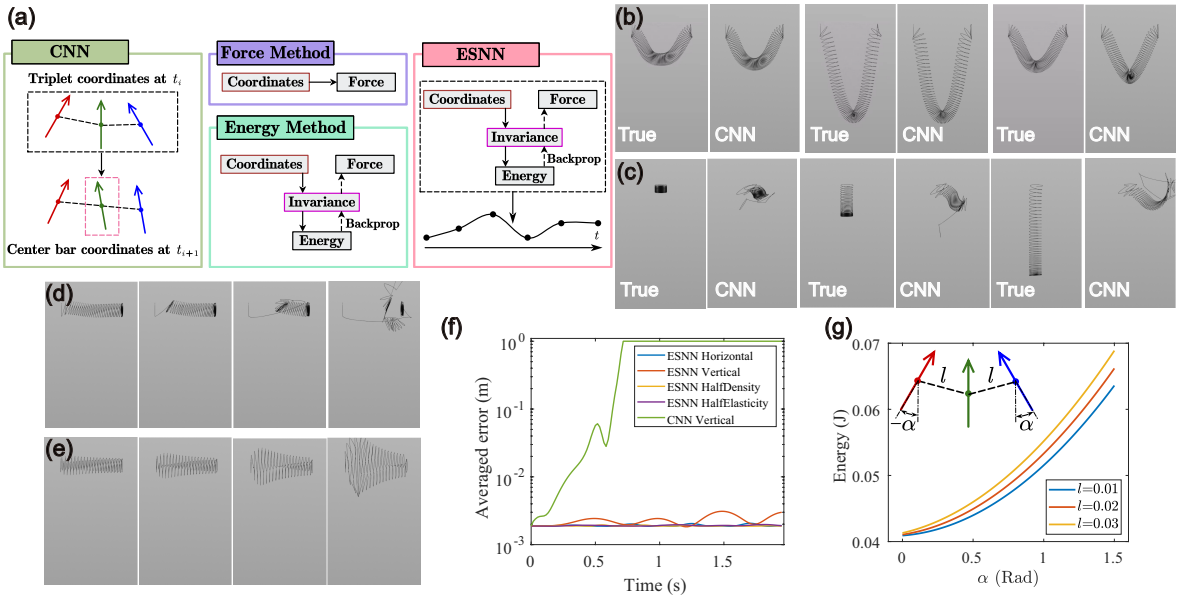


Figure 3.14: Comparison between the ESNN and 3 other methods (CNN, direct force predictor, direct energy predictor). (a) Schematic comparison between different approaches. CNN takes the triplet coordinates at the current time step and predicts the center bar coordinates at the next time step without any physical symmetry or dynamics encoded. The direct force predictor method has the same input but predicts the forces on the center bar. The direct energy predictor method incorporates invariances to predict the energy first and then by backpropagation derives an equivariant force field. Both of the "direct predictor" methods perform data-fitting directly. The ESNN trains an invariant NN under the NODE framework. (b) Comparison between 3D DER ground truth and CNN training results. Time shots are at 0.20 s, 0.43 s, and 1.18 s (from left to right). (c) Comparison between 3D DER ground truth and CNN generalization results. Time shots are at 0.03 s, 0.17 s, and 0.33 s (from left to right). (d) The training results of the direct force predictor method at 9.4 ms, 11.4 ms, 13.4 ms, and 14.9 ms (from left to right). The direct force predictor method fails to generate a reasonable simulation. (e) The training results of the direct energy predictor method at 7.5 ms, 8.5 ms, 9.5 ms, and 10.5 ms. The simulation diverges at 10.7 ms. (f) The error comparison between the ESNN (4 generalization cases) and CNN (generalization for a 40-cycle Slinky). The error is the vertex distance between NN prediction and 3D DER averaged on all the Slinky vertices. The CNN generalization error is significantly larger than those of the ESNN (capped at 1.0 for visualization). (g) The learned surrogate energy of the ESNN at different bending angles α. The symmetric rotation configuration in the inlet ensures that at different bending angles, the stretching stays the same and the shear remains 0. The energy increase is purely caused by bending motion. The bending energy and the associated bending moment are different for the same bending angle under different stretchings. This indicates a nonlinear coupling between stretching and bending.

CHAPTER 4

Interpretable Deep Meta-learning of Families of Dynamical Systems

4.1 Introduction

Chapter 3 has demonstrated the power of data-driven modeling of dynamical systems. With properly designed inductive bias that incorporates physical prior knowledge, a data-driven model can generalize to many unseen cases. However, the applicability and data-efficiency of such data-driven methods remain restricted, primarily due to the necessity of fitting distinct models for each system instance with different parameters, although they share the same dynamical structure.

Even if two systems share the form of dynamics, varied physical parameters lead to distinctive behaviors. Therefore, most previous works fit dedicated models separately for different system instances, which limits the models' applicability to one specific system. One solution is to include the systems' physical parameters as a part of the neural network model input [LP21, DMJ22]. This method, however, requires non-trivial expert knowledge to identify the characteristic system parameters, which is often difficult for complex or novel systems that necessitate data-driven modeling in the first place. Recently [ZWT18] considered systems comprising multiple interacting objects and attempted to learn one shared model for object motion and their interaction with the object properties unlabelled. This approach, however, does not apply to most systems where system-level dynamics are tracked.

For system instances that share the same dynamical structure but differ in parameters, learning separate models does not optimally utilize data, which leads to lower data efficiency.

In this chapter, our goal is to learn the shared dynamics form from the trajectories generated by a series of dynamical system instances despite their diversified behaviors in data, *without labels of system parameters*. The learned dynamics form can subsequently boost the data-driven modeling of new systems that share the dynamics form. This goal aligns well with that of multi-task meta-learning [WZL21], which aims to leverage the similarities between different tasks to enable better generalization and efficient adaptation to both seen and unseen tasks. In particular, a line of works is termed gradient-based meta-learning (GBML) [FAL17, NAS18, FRK19, RFK19, RRB19], which, given a set of tasks and a neural network model, tries to find an initialization of the model such that the adaptation to a task takes only a small number of gradient descent steps while resulting in good generalization on the task's test data.

We propose an efficient and interpretable method to model a family of dynamical systems using their observed trajectories, by combining GBML with neural ordinary differential equations introduced in Chapter 3. The method generalizes well on unseen systems from the same family, and the adaptation parameters show good interpretability. The intrinsic dimension of the varying system parameters can be estimated by analyzing the adaptation parameters. Given the ground truth of the system parameters, simple correspondence can be established between the adaptation parameters and actual physical parameters through diffeomorphism, which can be utilized as a "*Neural Gauge*" to measure the properties of new systems through observed trajectories. We name our method **i**nterpretable **M**eta **N**eural **ODE** (iMODE).

4.2 Problem Statement and Data Schema

Similar to Chapter 3, we consider autonomous dynamical systems described by ODE

$$\dot{\boldsymbol{w}}(t) = \frac{\mathrm{d}\boldsymbol{w}(t)}{\mathrm{d}t} = \boldsymbol{f}_{\boldsymbol{\varphi}}(\boldsymbol{w}(t)) \tag{4.1}$$

where $\boldsymbol{w}(t)$ is the state of the system at time t, and $\boldsymbol{\varphi}$ is the system parameters that parameterize the temporal derivative of states $\boldsymbol{f}_{\boldsymbol{\varphi}}$, which prescribes the evolution of the state.

To clarify the terms, we take the simple pendulum system as an example. An instance is a pendulum with a specific arm length, therefore the system parameter $\boldsymbol{\varphi}$ is the arm length. For different instances, the functional form of $\boldsymbol{f}_{\boldsymbol{\varphi}}$ is the same while the value of $\boldsymbol{\varphi}$ is varied as the parameter of $\boldsymbol{f}_{\boldsymbol{\varphi}}$. The state variable contains the location and speed of the pendulum and a trajectory consists of the state observed during a certain amount of time.

In the context of dynamics modeling for autonomous systems, the foundational task is to learn from a set of observed system trajectories so that the trajectories with unseen initial / boundary conditions of the system can be accurately predicted. The modeling of distinct system instances characterized by varied system parameters is regarded as separate tasks. In this chapter, we continue using the Neural ODE framework as in Chapter 3. Therefore, for a task $\mathcal{T}^{(i)}$ modeling a system i, the aim is to use a dynamics model $\widehat{\boldsymbol{f}}^{(i)}$ to approximate its dynamics $\boldsymbol{f}_{\boldsymbol{\varphi}^{(i)}}$, where we use parenthesized superscripts to index the system instances. Data-wise, $\mathcal{T}^{(i)}$ involves a training set and an evaluation set of trajectories — the training set $\mathcal{D}^{(i),\text{train}} = \begin{bmatrix}\mathbf{T}_1^{(i)} & \dots & \mathbf{T}_K^{(i)}\end{bmatrix}$ and the evaluation set $\mathcal{D}^{(i),\text{eval}} = \begin{bmatrix}\mathbf{T}_{K+1}^{(i)} & \dots & \mathbf{T}_N^{(i)}\end{bmatrix}$, where K is the number of trajectories in the training set, and N is the total number of trajectories that are associated with the task. The modeling *quality* of a model $\widehat{\boldsymbol{f}}^{(i)}$ can be quantified by $\mathcal{L}(\widehat{\boldsymbol{f}}^{(i)}, \mathcal{D}^{(i),\text{test}})$, the prediction error on the evaluation set and the modeling *efficiency* by the time and amount of data required to obtain a model with a certain level of performance $\mathcal{L}$.

The goal is to leverage the shared dynamics form learned from multiple instances (corresponding to tasks $\mathcal{D}^{\text{meta-train}} = \{\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(N_\text{s})}\}$, where N_s denotes the number of system instances in the meta-training set), so that given a new system with an unseen value of $\boldsymbol{\varphi}$, its dynamics can be efficiently modeled with a minimal amount of data. In other words, using the knowledge extracted from $\mathcal{D}^{\text{meta-train}}$, to achieve low modeling loss L for test tasks from $\mathcal{D}^{\text{meta-test}} = \{\mathcal{T}^{(1)}, \dots, \mathcal{T}^{(N_\text{s})}\}$ with minimal data requirement and computational cost.

4.3 Parameter Partitioning for Dynamics Modeling and System Adaptation

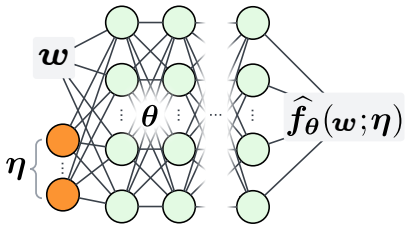


Figure 4.1: A neural module $\widehat{\boldsymbol{f}}$ parameterized by $\boldsymbol{\theta}$ takes the concatenation of system state $\boldsymbol{w}$ and the adaptation parameters $\boldsymbol{\eta}$ and generates the estimated force as output.

Recognizing that the system instances have shared dynamics form and varying physical parameters, we separate the neural network model parameters into two parts: the *shared parameters* $\boldsymbol{\theta}$ that capture the shared form of dynamics, i.e. the meta-knowledge, and the *adaptation parameters* $\boldsymbol{\eta}$ that account for variations across system instances.

A neural network $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{w};\boldsymbol{\eta})$ (Figure 4.1) replaces $\boldsymbol{f}_{\boldsymbol{\varphi}}(\boldsymbol{w})$ in Equation (4.1) to approximate the observed trajectories, where $\boldsymbol{\eta}$ is adapted to each system instance such that with a certain $\boldsymbol{\eta}^{(i)}$, $\widehat{\boldsymbol{f}}^{(i)} = \widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{w};\boldsymbol{\eta}^{(i)})$ approximates the dynamics of the i^{th} system instance. After training, $\boldsymbol{\eta}$ becomes a proxy for the physical parameters $\boldsymbol{\varphi}$. $\boldsymbol{\theta}$ is the model parameters that capture the functional form of dynamics shared across system instances. The predicted trajectory starting from an initial condition $\boldsymbol{w}_0 = \boldsymbol{w}(t_0)$ is given by integration (the 5th-order Dormand-Prince-Shampine solver is used to compute integrals)

$$\widehat{\boldsymbol{w}}(t, \boldsymbol{w}_0, \boldsymbol{\theta}, \boldsymbol{\eta}) = \boldsymbol{w}_0 + \int_{t_0}^{t} \widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}\left(\widehat{\boldsymbol{w}}(\tau);\boldsymbol{\eta}\right) \mathrm{d}\tau, \quad \widehat{\boldsymbol{w}}(t_0) = \boldsymbol{w}_0 \tag{4.2}$$

4.4 Bi-level Optimization for Joint Learning of a Family of Systems

Let us first define some notations for clarity. We represent the j^{th} trajectory of the i^{th} system instance as $\mathbf{T}_j^{(i)} = \left[\boldsymbol{w}^{(i)}(t_0), \ldots, \boldsymbol{w}^{(i)}(t_T)\right]$. The trajectory prediction generated by a neural module $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\,\cdot\,;\boldsymbol{\eta})$ given its initial state as in Equation (4.2), is denoted as $\widehat{\mathbf{T}}_j^{(i)}(\boldsymbol{\theta}, \boldsymbol{\eta})$.

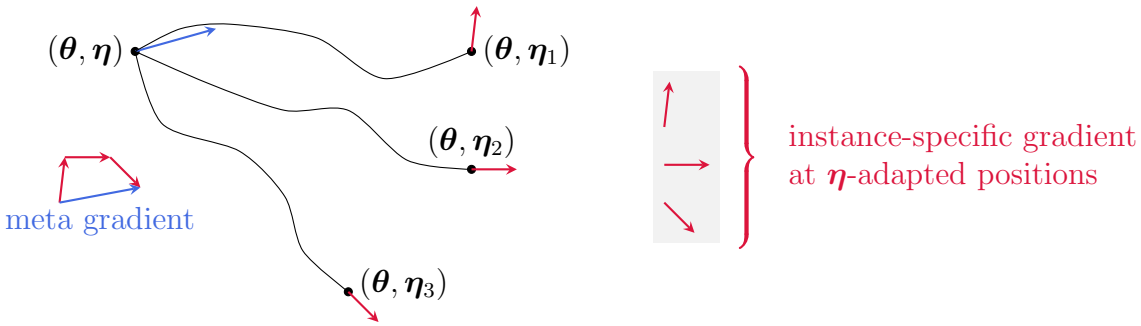


Figure 4.2: The bi-level optimization in the iMODE method. The NN weights $\boldsymbol{\theta}$ are shared across system instances while $\boldsymbol{\eta}$ is adapted for each instance. The meta gradient w.r.t. $\boldsymbol{\theta}$ aggregates the gradients evaluated with instance-adapted $\boldsymbol{\eta}$.

We define $\| \mathbf{T}_j^{(i)} - \widehat{\mathbf{T}}_j^{(i)}(\boldsymbol{\theta}, \boldsymbol{\eta}) \|^2$ to be the squared difference between $\mathbf{T}_j^{(i)}$ and $\widehat{\mathbf{T}}_j^{(i)}(\boldsymbol{\theta}, \boldsymbol{\eta})$ across all time steps, which mathematically expands to $\sum_t \left(\boldsymbol{w}_j^{(i)}(t) - \widehat{\boldsymbol{w}}^{(i)}(t, \boldsymbol{w}_j^{(i)}(t_0), \boldsymbol{\theta}, \boldsymbol{\eta}) \right)^2$.

The goal of the modeling is formulated as a bi-level optimization,

$$\textbf{outer:} \quad \min_{\boldsymbol{\theta}} \ \widetilde{\mathcal{L}}(\boldsymbol{\theta}) = \frac{1}{N_\mathrm{s}} \sum_{i=1}^{N_\mathrm{s}} \mathcal{L}^{(i)}(\boldsymbol{\theta}, \boldsymbol{\eta}_m^{(i)}), \text{ where} \tag{4.3}$$

$$\mathcal{L}^{(i)}(\boldsymbol{\theta}, \boldsymbol{\zeta}) = \frac{1}{K} \sum_{j=1}^{K} \| \mathbf{T}_j^{(i)} - \widehat{\mathbf{T}}_j^{(i)}(\boldsymbol{\theta}, \boldsymbol{\zeta}) \|^2, \tag{4.4}$$

$$\textbf{inner:} \quad \boldsymbol{\eta}_{l+1}^{(i)} = \boldsymbol{\eta}_l^{(i)} - \alpha \nabla_{\boldsymbol{\eta}} \mathcal{L}^{(i)}(\boldsymbol{\theta}, \boldsymbol{\eta}_l^{(i)}), \ \ \boldsymbol{\eta}_0^{(i)} = \boldsymbol{\eta} \tag{4.5}$$

where the inner-level involves an m-step gradient descent adapting $\boldsymbol{\eta}$ for each instance, while the outer-level finds the optimal initialization for $\boldsymbol{\theta}$. α is the inner-level stepsize and $\boldsymbol{\eta}_m^{(i)}$ is the adaptation parameters for the i^{th} system instance after m steps of adaptation. For short, we denote such i^{th} adaptation result as $\boldsymbol{\eta}^{(i)}$. Note that $\boldsymbol{\eta}^{(i)}$ depends on both $\boldsymbol{\theta}$ and $\boldsymbol{\eta}$ as shown in Equation (4.5). To avoid higher-order derivatives, we simplify such dependency following the first-order Model Agnostic Meta-Learning (first-order MAML) [FAL17] and take the outer-level step as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\beta}{N_\mathrm{s}} \sum_i \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\eta}^{(i)}), \ \text{(assuming that } \frac{\partial \boldsymbol{\eta}^{(i)}}{\partial \boldsymbol{\theta}} = \mathbf{0}) \tag{4.6}$$

where β is the outer-level stepsize. At both the inner-level and outer-level, the gradient

calculation for functions involving integrals is enabled by Neural ODE. The relation between gradient descent steps at the inner-level and the outer-level in the bi-level optimization is illustrated in Figure 4.2.

4.5 Dynamics Modeling for Dynamical System Families

In this section, we examine the modeling capability of the proposed iMODE method in terms of trajectory prediction *quality* on both the meta-training dataset and the meta-test dataset of unseen system instances, as well as the *efficiency* of such data-driven modeling. Various dynamical system families are considered, most of which are mechanical dynamical systems.

When it comes to mechanical dynamical systems, which are second-order autonomous, we again focus on Equation (3.1) in Section 3.3.1. For convenience, we repeat it below:

$$\dot{\boldsymbol{w}} = \begin{bmatrix} \dot{\boldsymbol{u}} \\ \ddot{\boldsymbol{u}} \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{u}} \\ \mathbf{M}^{-1}\ \boldsymbol{F}(\boldsymbol{w}) \end{bmatrix}, \quad \text{where } \boldsymbol{u} \text{ is the coordinates of the system}$$

In this case where the dynamics is essentially prescribed by a force field function $\boldsymbol{F}(\boldsymbol{w})$, the neural network module $\widehat{\boldsymbol{f}}$ is used to approximate $\boldsymbol{F}(\boldsymbol{w})$ instead of the whole $\dot{\boldsymbol{w}}$. Again, we assume normalized inertia $\mathbf{M} = \mathbf{I}$.

We demonstrate the modeling capabilities of the proposed method with three system families: simple pendulum oscillators, bistable oscillators, and Van der Pol oscillators. These three families cover a spectrum of complexity. A simple pendulum oscillator is conservative and can be characterized by a quadratic potential energy function with a single system parameter. Each of the conservative bistable oscillators we consider is characterized by a quartic potential energy function that has two system parameters and two local minima. The Van der Pol oscillators are non-conservative and each has three system parameters.

Using the modeling of Van der Pol systems as an example, Figure 4.3 shows that $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\cdot;\boldsymbol{\eta})$ specifies a force field that morphs as $\boldsymbol{\eta}$ changes. Trajectories with arbitrary initial conditions

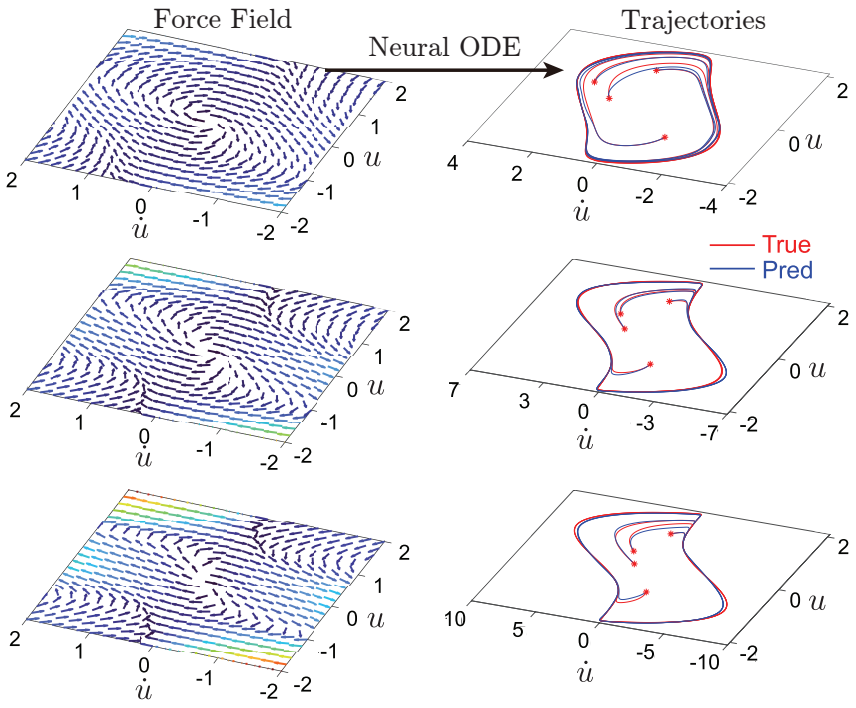


Figure 4.3: **Left**: Examples of estimated force field $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\,\cdot\,;\boldsymbol{\eta})$ for Van der Pol system instances that differ in their ϵ parameter (in ascending order from top to bottom). The estimation quality is further evaluated through the trajectories generated by the fields as shown on the right. **Right**: The estimated force field can be used to predict system trajectories for unseen initial conditions through integration (Equation (4.2)). The signature limit cycles of Van der Pol systems are faithfully reproduced.

can be predicted based on the force field using Neural ODE. See Section 4.5.1.3 for more details on Van der Pol systems.

4.5.1 Dynamics Modeling Accuracy

4.5.1.1 Simple Pendulum

The dynamics of a simple pendulum (Figure 4.4) can be described by the ODE Equation (4.7)

$$ml^2\ddot{\boldsymbol{u}} + mgl\sin(\boldsymbol{u}) = 0 \quad \text{s.t.} \quad \boldsymbol{u}(0) = \boldsymbol{u}_0,\ \dot{\boldsymbol{u}}(0) = \dot{\boldsymbol{u}}_0 \tag{4.7}$$

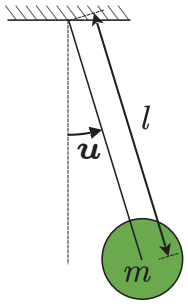
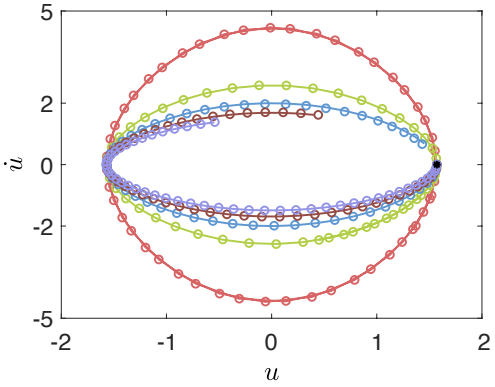


Figure 4.4: Illustration of the single pendulum system. The one-dimensional DoF is the angle between the pendulum arm and the vertically downward direction.

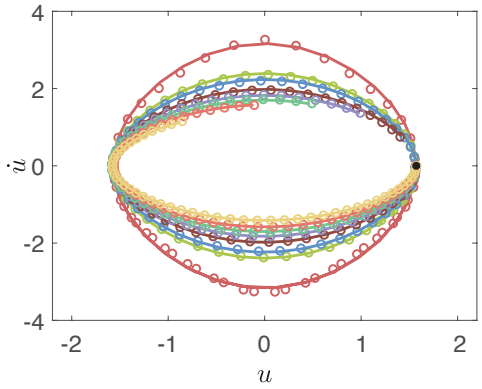
where the coordinate $\boldsymbol{u}$ of the system is simply the angle of the pendulum, and the state is the concatenation of the angle and its angular speed, making the state space a 2D plane. Being conservative in mechanical energy, the trajectories of a system instance in the state space is a closed curve.

The oscillating pendulum has 1 physical parameter, i.e. the arm length l (rotational inertia normalized). The training dataset contains 5 system instances with $l = 1, 3, 5, 7$, and 9 m. A long trajectory of 10 s is generated following Equation (4.7) for each instance with the initial position of $\pi/2$ and velocity of $0\,\mathrm{rad} \cdot \mathrm{s}^{-1}$ and simulation time step of 10 ms. During training, a batch of 20 1 s-long clips is sampled randomly in each epoch. Essentially the iMODE training sees 1 s-long trajectories with various initial conditions. The learned model is tested on 8 unseen system instances with $l = 2$, 3.5, 4, 5.1, 6, 6.9, 8, and 10 m. The adaptation to each unseen system uses similar data sampling as the training. For each testing system instance, a batch of 20 randomly sampled 1 s-long clips is fed to the neural network for adaptation of $\boldsymbol{\eta}$. The task adaptation only takes 5 steps. Then the adapted model for each instance is used to calculate a trajectory of 5 s given an initial condition, and compared with ground truth.

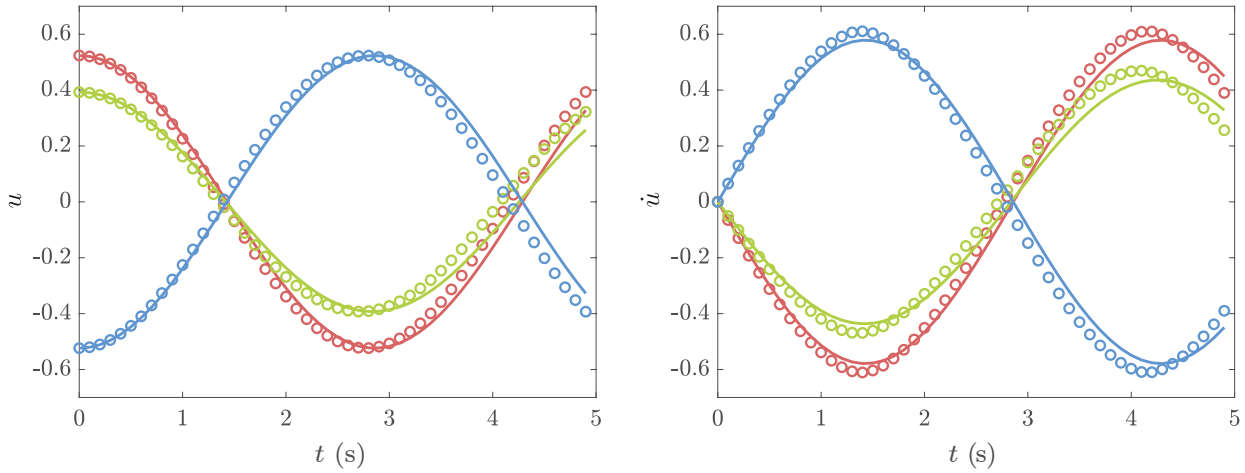
The results are shown in Figure 4.5. The solid lines (the ground truth) match well the circles (prediction), showing that iMODE successfully models the simple pendulum systems with reliable prediction accuracy.



(a) The predicted simple pendulum trajectories vs the ground truth in the meta-training dataset. Different colors correspond to different system instances.



(b) The predicted simple pendulum trajectories vs the ground truth in the meta-test dataset. Different colors correspond to different system instances.



(c) The plot of an adapted model's prediction of a simple pendulum ($l = 7\,\mathrm{m}$) system's coordinate vs time and velocity vs time with three unseen initial states. Different colors correspond to different initial conditions.

Figure 4.5: The meta-learning results for the simple pendulum systems. The iMODE trajectory prediction (circles) for different arm lengths l and initial conditions (different colors) match those of the ground truth (solid lines).

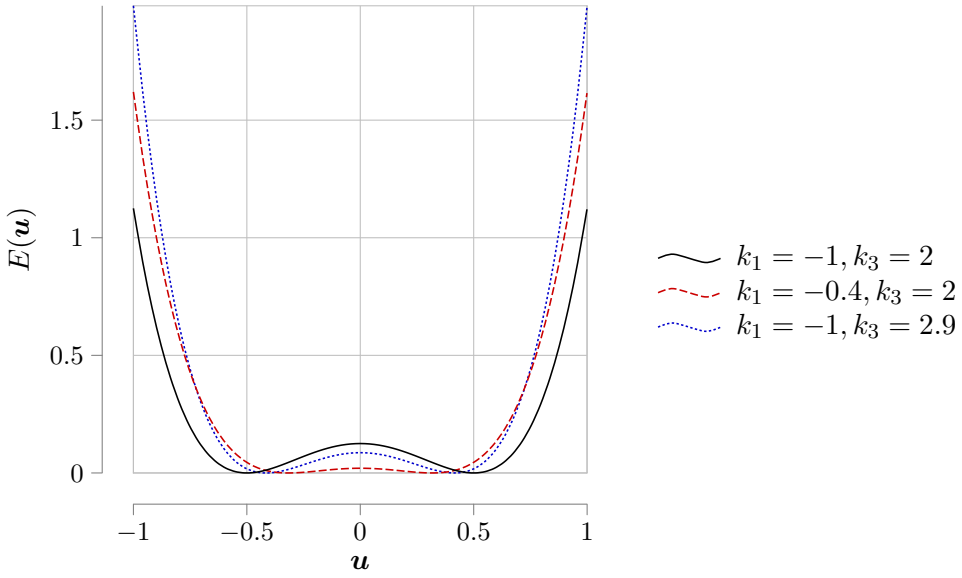


Figure 4.6: Examples of the quartic potential function of the bistable oscillator systems.

4.5.1.2 Bistable Oscillator

For a slightly more complicated case, we consider the system family described by ODE Equation (4.8) with a symmetric quartic potential $E(\boldsymbol{u}) = k_1\boldsymbol{u}^2 + k_3\boldsymbol{u}^4 + \text{const}$. Figure 4.6 illustrates some examples of such potential functions (with their minima shifted to zero by adjusting the constant term). Again, the DoF is 1 and the state is the concatenation of the only coordinate and its time derivative.

$$\ddot{\boldsymbol{u}} + k_1\boldsymbol{u} + k_3\boldsymbol{u}^3 = 0 \quad \text{s.t.} \quad \boldsymbol{u}(0) = x_0,\ \dot{\boldsymbol{u}}(0) = \dot{x}_0 \tag{4.8}$$

A bistable system has a potential energy function controlled by 2 parameters k_1 and k_3. The potential energy has two local minima, or so-called potential wells. When the initial conditions vary, the bistable system can oscillate intra-well or inter-well. Figure 4.7(a) shows that the task adapted trajectories ($m = 5$) match the ground truth well.

The training dataset contains 20 system instances, a mesh of $k_1 = -0.4$, -0.6, -0.8 and -1.0 and $k_3 = 2.0$, 2.9, 3.7, 4.6, and 5.0. Trajectories of multiple initial conditions with stepsize of 10 ms and duration of 10 s are generated with Equation (4.8) for each

instance. During training, a batch of 100 randomly sampled 1 s-long clips is used for each epoch. During testing, task adaptation takes 5 steps on previously unseen systems $[k_1, k_3] = [-0.5, 3.1], [-0.7, 4.2], [-0.5, 4.7]$. The learned models calculate trajectories of 5 s given an initial condition. The results are shown in Figure 4.7(b).

4.5.1.3 Van der Pol System

The Van der Pol systems described by Equation (4.9) have 1 DoF and 3 physical parameters $\boldsymbol{\varphi} = [\epsilon, \delta, \omega]$. They exhibit limit cycles due to the negative damping for small oscillation amplitudes. Figure 4.7(a) shows that the variation of limit cycles due to the change of physical parameters is well predicted.

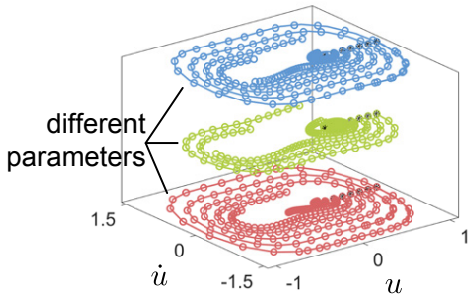
$$\ddot{\boldsymbol{u}} - \epsilon\dot{\boldsymbol{u}}(1 - \delta\boldsymbol{u}^2) + \omega^2\boldsymbol{u} = 0 \quad \text{s.t.} \quad \boldsymbol{u}(0) = x_0,\ \dot{\boldsymbol{u}}(0) = \dot{x}_0 \tag{4.9}$$

The training dataset contains 27 system instances, a mesh of $\epsilon = [1.0, 2.0, 3.0]$, $\delta = [1.0, 2.0, 3.0]$, and $\omega = [0.5, 1.0, 1.5]$. Trajectories of multiple initial conditions with stepsize of 10 ms and duration of 10 s are generated for each instance. During training, a batch of 100 randomly sampled 1 s-long clips is used for each epoch. During testing, task adaptation takes 5 steps on previously unseen systems instances $[\epsilon, \delta, \omega] = [1.2, 1.2, 2.1], [1.2, 1.8, 1.4], [2.6, 1.5, 2.5]$. The learned models calculate trajectories of 5 s given an initial condition. The results are shown in Figure 4.8(b).

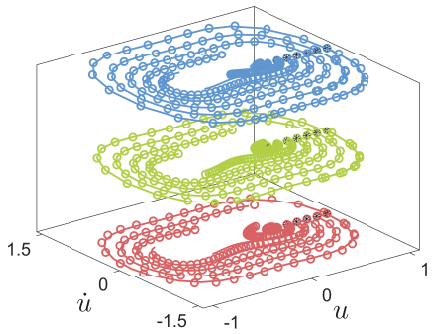
4.5.2 Dynamics Modeling Efficiency for New System Instances

Note that the m in Equation (4.5) is chosen to be quite small (e.g. 5), so given trajectories of a previously unseen system, $\boldsymbol{\eta}$ can be efficiently updated with few gradient descent steps, adapting the network to account for behaviors of the new system, which is one order of magnitude faster compared to training from scratch.

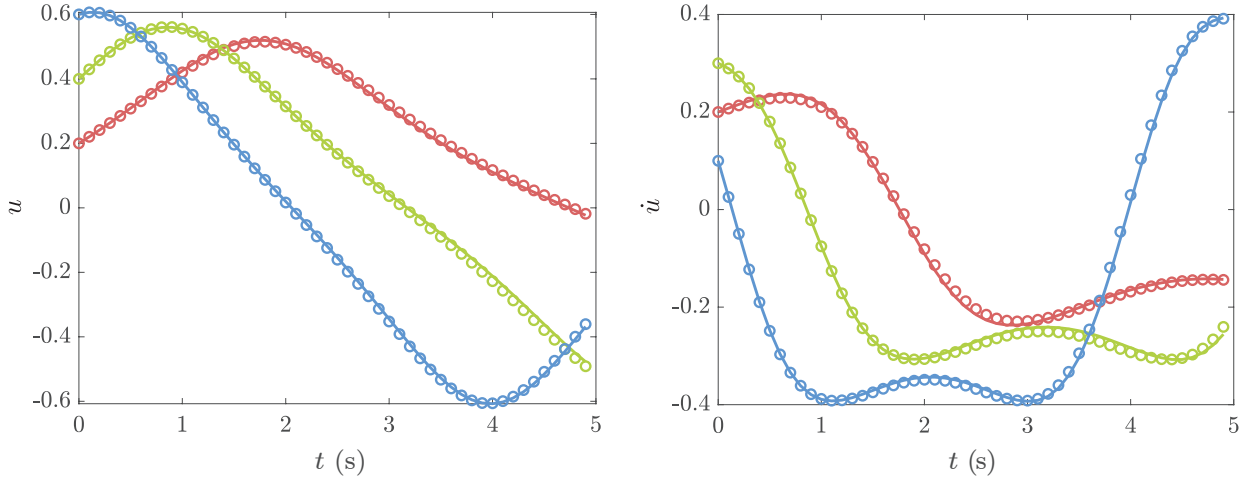
The fast adaptation of iMODE is demonstrated with the bistable systems in Figure 4.9. iMODE is able to adjust the (only two-dimensional) adaptation parameters in 5 steps to learn



(a) The predicted bistable oscillator trajectories vs the ground truth in the meta-training dataset (3 instances are picked for visualization). Different colors correspond to different system instances.

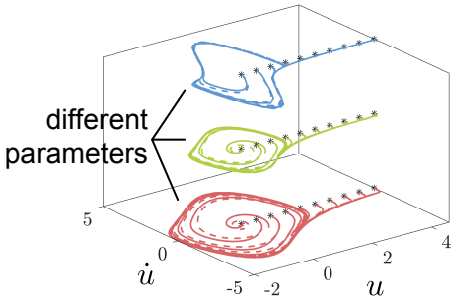


(b) The predicted bistable oscillator trajectories vs the ground truth in the meta-test dataset (3 instances are picked for visualization). Different colors correspond to different system instances.

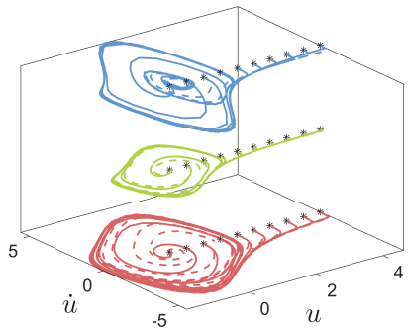


(c) The plot of an adapted model's prediction of a bistable oscillator ($k_1 = -0.6, k_3 = 5$) system's coordinate vs time and velocity vs time with three unseen initial states. Different colors correspond to different initial conditions.

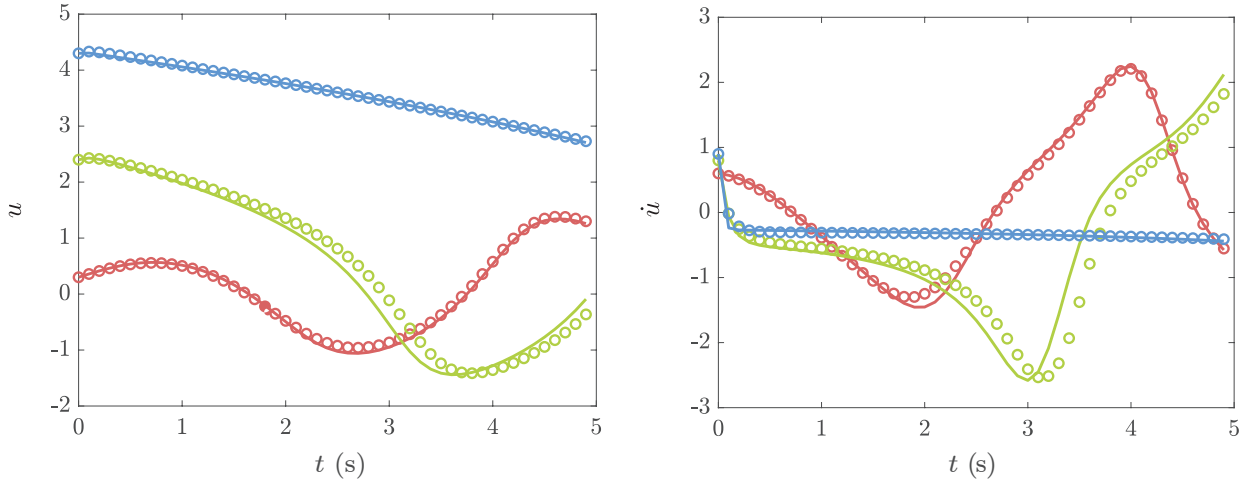
Figure 4.7: The meta-learning results for the bistable oscillators. The iMODE trajectory prediction (circles) for different arm lengths and initial conditions (different colors) match those of the ground truth (solid lines).



(a) The predicted Van der Pol oscillator trajectories vs the ground truth in the meta-training dataset (3 instances are picked for visualization). Different colors correspond to different system instances.



(b) The predicted Van der Pol oscillator trajectories vs the ground truth in the meta-test dataset (3 instances are picked for visualization). Different colors correspond to different system instances.



(c) The plot of an adapted model's prediction of a Van der Pol oscillator ($\epsilon = 1, \delta = 2, \omega = 1.5$) system's coordinate vs time and velocity vs time with three unseen initial states. Different colors correspond to different initial conditions.

Figure 4.8: The meta-learning results for Van der Pol systems. The iMODE trajectory prediction (circles) for different arm lengths and initial conditions (different colors) match those of the ground truth (solid lines).

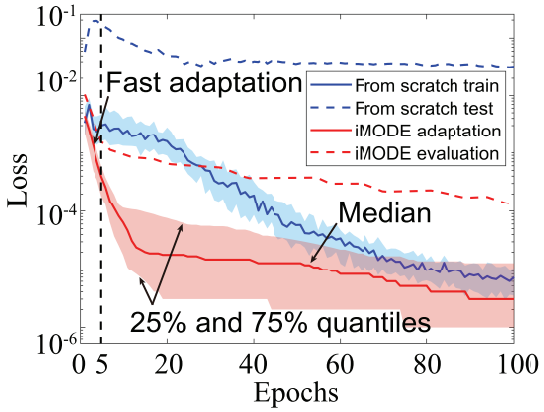


Figure 4.9: Comparison of iMODE adaptation v.s. training from scratch in terms of the loss decrease along training. The statistics are obtained by repeating experiments on 50 unseen bistable system instances with randomly chosen physical parameters. iMODE demonstrates fast adaptation and good generalization within the first 5 adaptation steps.

the dynamics of unseen system instances. Training the same network from scratch (random initialization) on the same test dataset requires much more epochs (and computation cost per epoch because all neural network parameters are updated) to achieve comparable accuracy. When evaluated on trajectories with unseen initial conditions (of distinct conserved energies), the performance of iMODE-adapted models outperforms that of the model trained from scratch by several orders of magnitude, showing superior generalization ability with limited data.

Further investigation verifies our thesis that iMODE achieves the aforementioned data-efficiency by successfully learning the shared dynamics form. To account for the conservative nature of the bistable system, we take a specialized form of the neural force estimator $\widehat{\boldsymbol{f}}$, similar to that of the ESNN in Chapter 3:

$$\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{u};\boldsymbol{\eta}) = \frac{\partial E_{\boldsymbol{\theta}}(\boldsymbol{u};\boldsymbol{\eta})}{\partial \boldsymbol{u}} \tag{4.10}$$

That is, the neural network first outputs a scalar energy and the force prediction is induced from the energy as the derivative w.r.t. the system coordinates. In this way, iMODE enables the fast adaptation of not only the force field but also the potential energy field for conservative dynamical systems. This also allows us to examine the "meta-knowledge"

learned by iMODE from a family of dynamical systems by inspecting the learned energy function, which can be much more easily visualized and understood compared to the vector function of the force field.

Figure 4.10(a) shows that the energy function learned by iMODE for each system instance in the meta-training dataset matches the ground truth. With only the adaptation parameter $\boldsymbol{\eta}$ varied, the neural energy estimator gives potential energy functions with varied landscapes, all of which have two symmetric local minima. This verifies that the iMODE learns the shared potential energy landscape signature — the double potential well — from multiple instances in the meta-training dataset.

When adapted to a single trajectory generated by an unseen system instance, iMODE keeps the landscape signature in its predicted energy function, which leads to good performance even if the data from the new system is scarce. Figure 4.10(b) demonstrates one example of such data-efficient modeling of a new bistable system, of which only a trajectory of intra-well oscillation is given. In this case, the information contained in the provided trajectory is insufficient to depict the entire potential energy landscape. The energy function estimated by the adapted iMODE network is very close to the true energy function, while a neural network of the same architecture and the same number of parameters, trained from scratch on the trajectory, gives a totally incorrect energy estimation, and of course, has poor generalization performance for unseen initial conditions of the system.

4.6 Adaptation Parameter Interpretability

In iMODE, $\boldsymbol{\eta}$ modulates the neural network and makes $\widehat{\boldsymbol{f}}_{\boldsymbol{\theta}}(\boldsymbol{w}, \boldsymbol{\eta}^{(i)})$ the approximator of the i^{th} system's actual dynamics $\boldsymbol{f}_{\boldsymbol{\varphi}^{(i)}}$, making $\boldsymbol{\eta}$ a proxy for the physical parameters $\boldsymbol{\varphi}$ that can be leveraged to reveal information on the physical parameters when they are not available.

In this section, we first examine the correlation between adapted $\boldsymbol{\eta}$ and the actual physical parameters $\boldsymbol{\varphi}$ to confirm their correspondence. Then we demonstrate that the intrinsic

dimension of $\boldsymbol{\varphi}$ can be unveiled without supervision through principal component analysis (PCA) as an example of data-driven study of physical systems with the help of iMODE.

4.6.1 Correlation with Physical Parameters of System Instances

Figure 4.11 shows the correlation between adapted $\boldsymbol{\eta}$ and the physical parameters $\boldsymbol{\varphi}$ of the corresponding system instances by plotting the adapted $\boldsymbol{\eta}$ values in the space of $\boldsymbol{\eta}$ and annotating how the variation of $\boldsymbol{\varphi}$ aligns with the "constellations" of the $\boldsymbol{\eta}$ value points.

Such correlation can be used to study whether a controllable experiment parameter in $\boldsymbol{\varphi}$ influences the system dynamics by checking whether there is a component in the $\boldsymbol{\eta}$ space correlating with that controllable experiment parameter.

4.6.2 Inferring Intrinsic Dimension with PCA

The intrinsic dimension $d_{\boldsymbol{\varphi}}$ of the physical parameters $\boldsymbol{\varphi}$ can be estimated by applying Principal Component Analysis (PCA) to the collection of the $\boldsymbol{\eta}$ vectors, each adapted to one of the system instances. Using an "elbow" method on the cumulative explained variance ratio curve of the PCA result, the number of the principal components that explain most of the variance has a good correspondence with d_ϕ, as long as $d_{\boldsymbol{\eta}} \geq d_{\boldsymbol{\varphi}}$, where $d_{\boldsymbol{\eta}}$ is the dimension chosen for $\boldsymbol{\eta}$. The PCA results on the pendulum, bistable system, and Van der Pol system are shown in Figure 4.12. Taking the Van der Pol system as an example, $d_{\boldsymbol{\eta}}$ is respectively 3, 4, or 5 for the three curves with triangle markers. In all three cases, the first three principal components explain more than 99% of the variance, and the "elbow" appears at 3, which corresponds well with the fact that $d_{\boldsymbol{\varphi}} = 3$ for the Van der Pol system.

This technique can be useful to infer the number of "intrinsic" parameters instead of "apparent" controllable parameters of dynamical systems. For example, the dynamics of an axially vibrating rod is governed by E/ρ, the ratio between Young's modulus and the density of the material of the rod. In experiments, one can control both E and ρ (by changing the material), but the dimension of the "intrinsic" parameter is just 1, which can be revealed by

the optimal dimension of the adaptation parameter.

4.7 Application: Neural Gauge

Without labels for the physical parameters, iMODE develops a latent space of adaptation parameters accounting for the variations in dynamics among system instances. Given the physical parameter labels of the system instances in the training data, a mapping between the space of the physical parameters and the latent space can be established so that the corresponding physical parameters can be estimated given any point in the latent space. iMODE therefore can be exploited as a "Neural Gauge" to identify the physical parameters of unseen system instances, and the establishment of such mappings can be seen as a calibration process. We propose to construct such mappings as diffeomorphism, which can be learned with a neural ODE $\mathrm{d}\boldsymbol{z}(t)/\mathrm{d}t = \boldsymbol{g}_{\boldsymbol{\xi}}(\boldsymbol{z})$, such that starting from a given point in the latent space, $\boldsymbol{z}(0) = \boldsymbol{\eta}^{(i)}$, the state $\boldsymbol{z}$ at $t = 1$ gives the corresponding physical parameters, $\boldsymbol{z}(1) = \boldsymbol{\varphi}^{(i)}$, $i = 1, \ldots, N_\mathrm{s}$. For simplicity, the dimension of the latent space and that of the physical parameter space are assumed to match ($d_{\boldsymbol{\eta}} = d_{\boldsymbol{\varphi}}$). In general, such diffeomorphism can be established as long as $d_{\boldsymbol{\eta}} \geq d_{\boldsymbol{\varphi}}$ by padding zeros to the end state of $\boldsymbol{z}$ and let $\boldsymbol{z}(1) = \begin{bmatrix} \boldsymbol{\phi}_i^\mathrm{T} & 0 & \ldots & 0 \end{bmatrix}^\mathrm{T}$. $\boldsymbol{g}_{\boldsymbol{\xi}}$ is a NN whose weights are optimized by $\boldsymbol{\xi} = \arg\min_{\boldsymbol{\xi}} \sum_i \left\| \mathbf{z}^{(i)}(1) - \boldsymbol{\varphi}^{(i)} \right\|_2^2$.

Figure 4.13 shows the learned diffeomorphism for the bistable system. The diffeomorphism establishes a bijection between the physical space and the latent space of $\boldsymbol{\eta}$ so that a grid in the physical parameter space can be continuously transformed into the adaptation parameter space. The visualization highlights the advantages of diffeomorphism mapping: (1) The transformation is smooth so that the local geometric structure is preserved; (2) Invertible transformation allows a better interpretation of the latent space compared to degenerating ones.

After constructing the diffeomorphism, we test the physical parameter identification performance on 100 randomly selected unseen instances (with random physical parameters).

The identification errors are shown in Figure 4.14 for the simple pendulum, bistable, and Van der Pol systems. The end-to-end identification starting from data feeding normally takes around 2 seconds.

4.8 Complex Systems

In this section, we demonstrate iMODE on two more complex systems, the reduced-order Slinky system introduced in Chapter 3, and reaction-diffusion systems described by the Kolmogorov-Petrovsky-Piskunov (KPP) equation.

4.8.1 iMODE Applied to 2D Reduced-order Slinkies

In the Slinky case, we follow Chapter 3 to enforce Euclidean invariance of the energy field and induce equivariance of the force field. iMODE is able to learn from 4 Slinky cases (of Young's modulus 50, 60, 70, and 80 GPa, dropping under gravity from a horizontal initial configuration with both ends fixed) and then quickly generalize to an unseen Slinky under unseen initial and boundary conditions (Figure 4.15).

In the training set, the Slinkies are clamped at both ends and freely drop under gravity from a horizontal initial configuration. Two inner steps are taken to update $\eta \in \mathbb{R}$ for each Slinky. After training the NN, we perform task adaptation (2 steps) on a unseen Slinky of Young's modulus 56 GPa and observe a good fitting result (Figure 4.15(a)). The resulting NN is then directly applied to computation under an unseen boundary condition and Slinky orientation (the bottom end is free and the Slinky drops under gravity from a vertical initial configuration) without any modification (Figure 4.15(b)).

4.8.2 iMODE Applied to KPP systems

To solve the KPP equation, we discretize the spatial domain $[0, 1]$ into 20 segments. So the the partial differential equation system (here x denotes the spatial coordinate)

$$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ru(1-u) \tag{4.11}$$

is represented by an ordinary differential equation system containing 21 variables. The diffusion term is approximated by 2nd-order central difference and the diffusivity is assumed known. The meta-learning is performed to learn the reaction term with different reaction strength coefficients r (without knowing the mathematical form). The training dataset contains 5 systems with $r = 0.01,\ 0.02,\ 0.03,\ 0.04,\ 0.05$. The Neumann boundary condition $u'(0) = u'(1) = 0$ ($'$ denotes derivative with respect to x) is used across the training dataset. The iMODE task adaptation takes 5 iterations. The training results for $r = 0.01,\ 0.05$ are shown in Figure 4.16(a). The iMODE NN is then adapted on the data from a unseen system instance with $r = 0.034$. The resulted NN is directly applied to computation with unseen initial and boundary conditions (Dirichlet type $u(0) = u(1) = 1$). The results of the latter are shown in Figure 4.16(b) and a good agreement is observed. This again validates the capability of the iMODE algorithm to fast adapt on unseen complex parametric systems and accurately predict on initial and boundary conditions different from those in the training dataset.

Another testing result for the KPP system is shown in Figure 4.16(c). The testing has the same type of boundary condition ($u'(0) = u'(1) = 0$) as the training dataset but an unseen initial condition. The prediction (right) matches the ground truth (left) well.

4.9 Conclusion

We have presented the iMODE, the **i**nterpretable **M**eta **N**eural **ODE**, introducing meta-learning to dynamical system modeling. As a major difference from existing NN-

based methods, iMODE learns the functional form of the dynamics, from only observed trajectories from a family of dynamical systems, requiring no knowledge of the systems' physical parameters. The modeling capacity of iMODE, including prediction accuracy, efficiency, generalizability, and interpretability are demonstrated with various dynamical systems. iMODE could open new possibilities for numerous potential applications. To name a few, iMODE can be used to:

(a) Construct a mapping between the true physical parameters and the "apparent" controllable physical parameters (usually with significantly different physical meanings). For example, consider the case where one wants to study the physical parameters (and the number of them) that are responsible for force-deformation of biological cells. Such causal physical parameters are not directly controllable but can be significantly influenced by the concentrations of different ions in the culture medium. iMODE can be adopted to construct the mapping between the hidden parameters $\boldsymbol{\eta}$ (as surrogates of true physical parameters) and the concentrations of ions $\boldsymbol{\varphi}$.

(b) Determine whether a controllable experiment parameter $\boldsymbol{\varphi}$ influences the system dynamics by determining whether there is a component in the latent space correlating with that controllable experiment parameter. This is the causality analysis of $\boldsymbol{\varphi}$.

(c) Determine the number of "intrinsic" parameters instead of "apparent" controllable parameters.

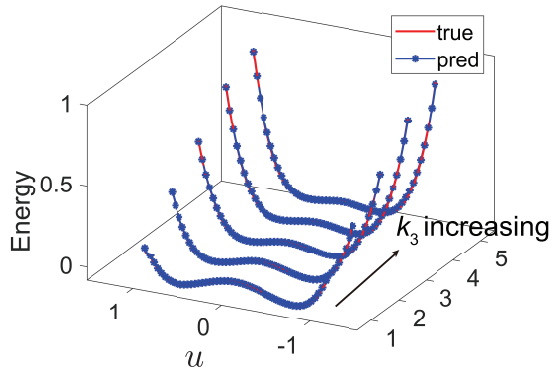
In particular, for reduced-order modeling, the "intrinsic" physical parameters of the reduced-order models may not be obviously and directly controllable in experiments.

(d) Provide latent encodings for parameters that are hard to characterize. One case is characterizing the fluid dynamics around an immersed bluff body conditioned on different geometries of the body, e.g. constructing a latent encoding space using iMODE for circle, square, star, and other irregular bluff body shapes. Once such shape encoding space is constructed, we can consult the iMODE model for dynamics of unseen bluff body shapes by adaptation in the encoding space.

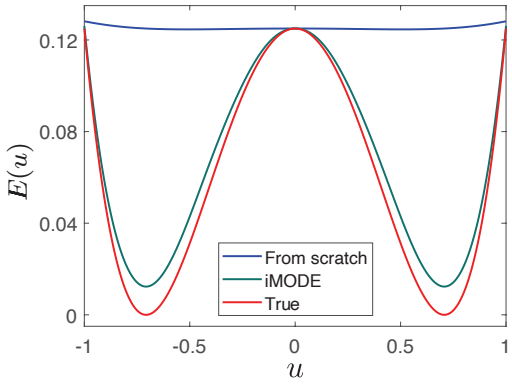
(e) To measure the physical parameters of a new system as a "Neural Gauge" if the system parameters $\boldsymbol{\varphi}$ are available. The "Neural Gauge" functionality of iMODE serves as a *virtual instrument* to measure the physical parameters of a new system by observing trajectories from the system.

4.10 Additional Information

This research was collaboratively conducted with Dr. Qiaofeng Li during his post-doctoral appointment in the Structures-Computer Interaction group at UCLA led by Prof. Khalid Jawed.

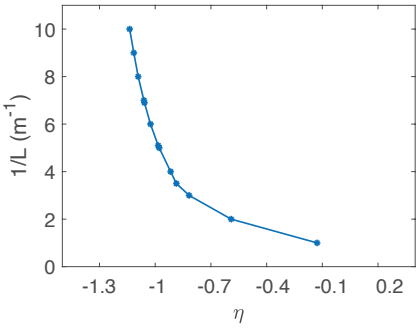


(a) The energy functions learned by iMODE for bistable oscillators in the meta-training set vs the ground truth.

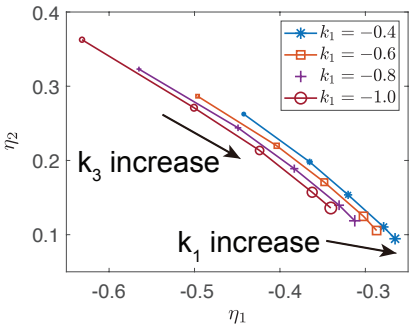


(b) The energy function given by iMODE adapted to a new bistable oscillator, the energy function given by a neural network trained from scratch on the new system only, and the ground truth.

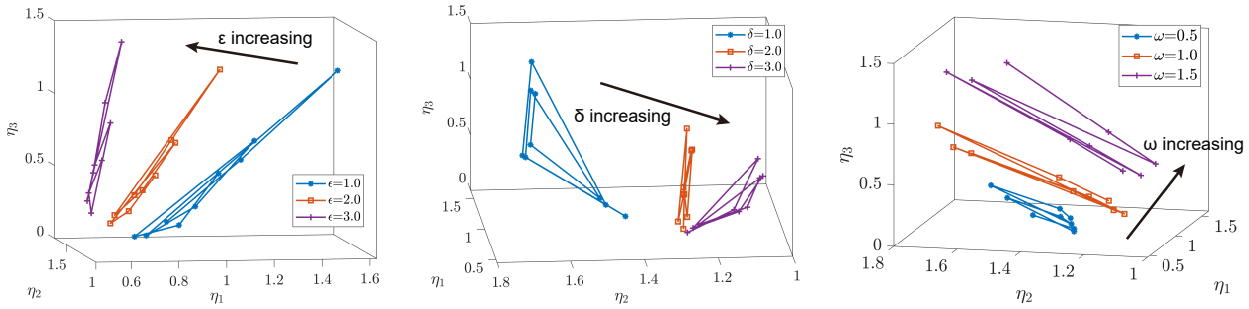
Figure 4.10: iMODE successfully learns the double-well potential energy of bistable oscillators.



(a) The learned $\boldsymbol{\eta}$ is in good correlation with the effective stiffness of different pendulums $(1/L)$.



(b) Two principal axes can be identified from the latent space of the learned $\boldsymbol{\eta}$, each regarding the variation of one physical parameter.



(c) The three variation directions in the latent space for the physical parameters of the Van der Pol system ϵ, δ, and ω.

Figure 4.11: The correspondence between $\boldsymbol{\eta}$ and $\boldsymbol{\varphi}$ in various system families.

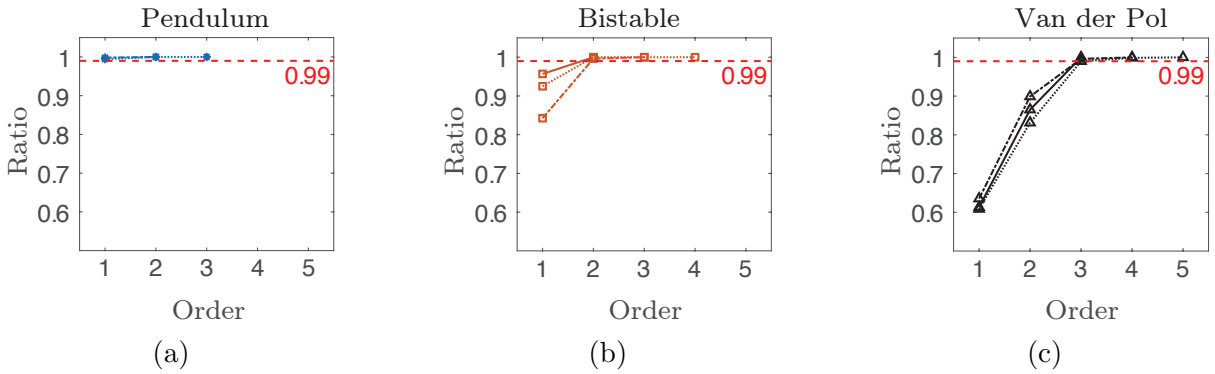


Figure 4.12: Inferring intrinsic dimension with PCA. The number of top PCA components that preserve a significant portion ($> 99\%$) of the variance gives a good estimation of the intrinsic dimension of true physical parameters.

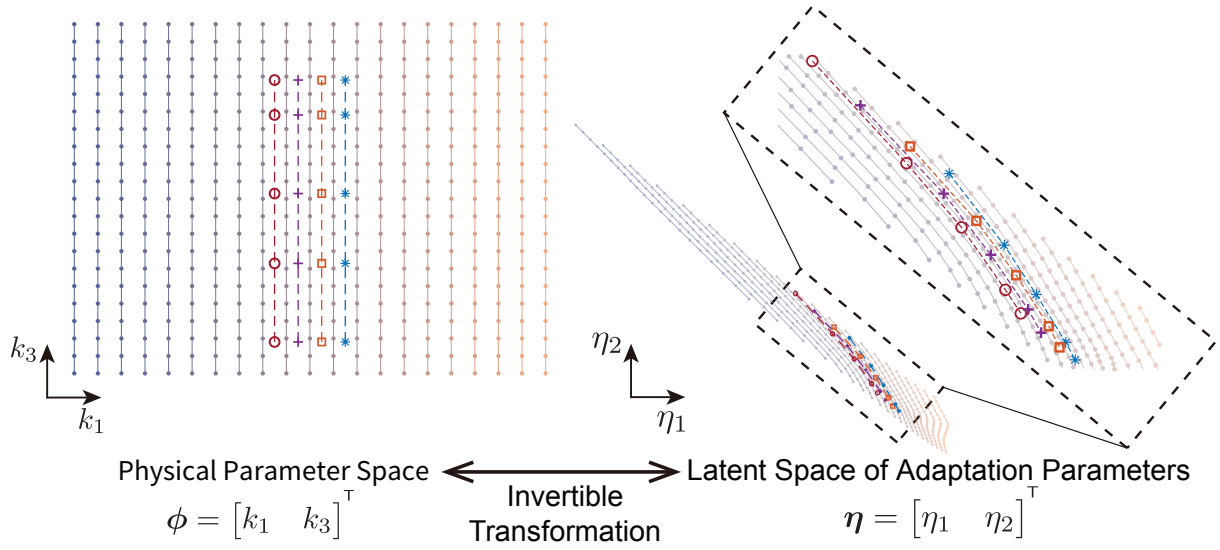


Figure 4.13: The diffeomorphism constructed for the bistable system. It shows how a grid in the physical space is continuously deformed into the latent space of adaptation parameters.

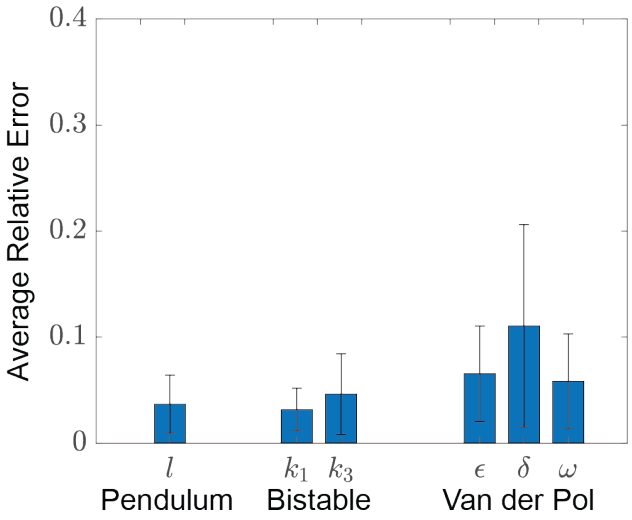


Figure 4.14: The mean error and computation time of *Neural Gauge* for 100 systems with randomly generated unseen parameters.

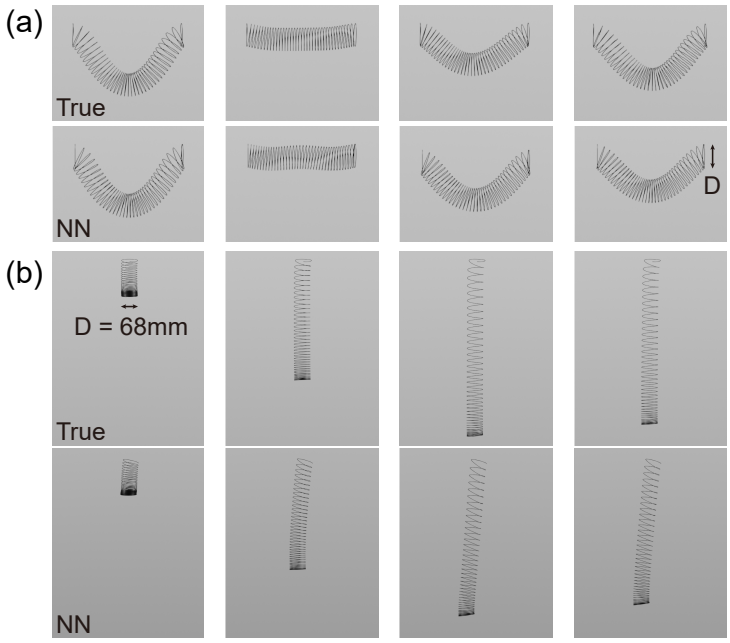


Figure 4.15: iMODE applied to Slinky. (a) The testing performance of the iMODE model on an unseen Slinky (of an unseen Young's modulus) with the same boundary condition as the training dataset. The top row is the ground truth and the bottom row is the iMODE model prediction at 0.28 s, 0.47 s, 0.65 s and 0.83 s (left to right). The mean squared error of the 3D Slinky reconstruction over the entire trajectory is $8 \times 10^{-4}\,\mathrm{m}^2$. (b) The testing performance of the iMODE model on unseen initial and boundary conditions. The top row is the ground truth and the bottom row is the iMODE model prediction at 0.15 s, 0.32 s, 0.48 s and 0.65 s (left to right). The mean squared error of the 3D Slinky reconstruction over the entire trajectory is $1.26 \times 10^{-3}\,\mathrm{m}^2$.

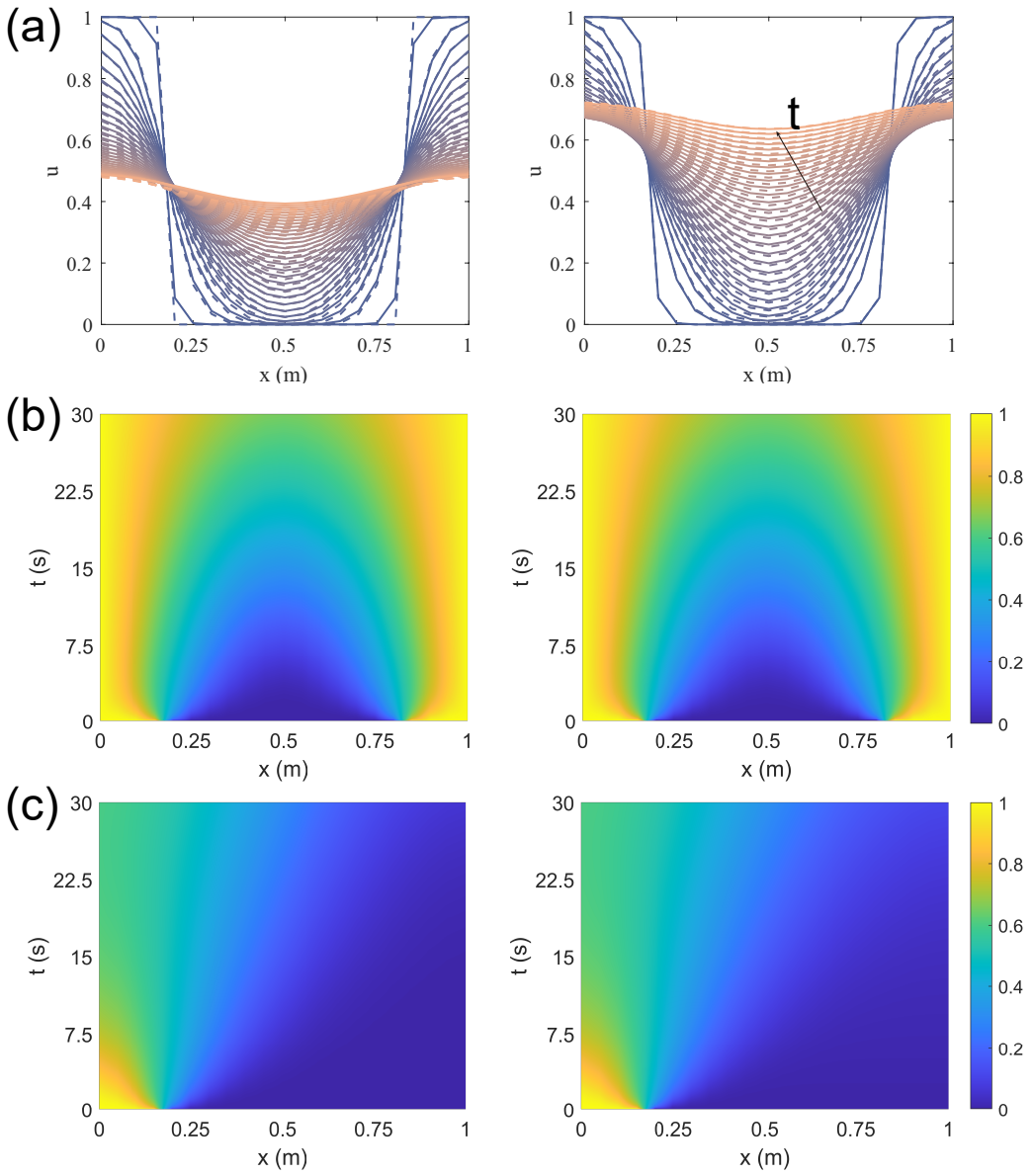


Figure 4.16: iMODE applied to KPP systems. (a) The training results of the iMODE algorithm on the KPP system for $r = 0.01$ (left) and $r = 0.05$ (right). Solid lines are ground truth. Dashed lines are iMODE predictions. The arrow indicates time marching of u. (b) The iMODE testing results on an unseen system $r = 0.034$ with an unseen boundary condition. The ground truth (left) match the iMODE prediction (right) well. (c) The iMODE testing results ($r = 0.034$) on an unseen initial condition from the training dataset. The ground truth (left) match the iMODE prediction (right) well.

CHAPTER 5

Conclusions and Future Research Opportunities

In this chapter, we conclude the dissertation and propose the problems to be studied in the future following this dissertation.

5.1 Conclusions

Deep neural networks, as a powerful data-fitting tool, have unlocked a new realm of modeling possibilities, capable of handling vast amounts of data across numerous disciplines. These tools are now not only able to contend with previously unmanageable complexity of data and problems, but also to facilitate knowledge discovery. In an optimistic vision, such techniques pave the way to a future where artificial intelligence agents learn the knowledge of the world effectively and build predictive models without human supervision, thereby enabling revolutionary automation.

Describing how the state of a system evolves over time, the study of the ubiquitous dynamical systems provides a crucial lens to understand the world. We have demonstrated ways to address unique challenges faced by the data-driven modeling approach based on deep learning, in particular, its low data efficiency compared to classic approaches.

First, we recognize the importance of physics prior knowledge and design a deep learning framework that incorporates physics principles including Euclidean symmetry and Newton's second law. The framework combines a novel network architecture that guarantees Euclidean equivariance of the estimated elastic force and Neural ODE [CRB18]. The method achieves unparalleled data efficiency and generalization ability, accurately predicting trajectories for unseen initial / boundary conditions while trained on only a single demonstration trajectory

of a Slinky's motion.

Second, we draw attention to the limitation of the majority of current data-driven modeling methods that they require knowledge of the system parameters to make use of the data from multiple dynamical system instances from the same family and learn the shared form of dynamics. We accordingly propose **i**nterpretable **M**eta **N**eural **ODE** (iMODE) that learns the common dynamics form shared by multiple dynamical system instances without requiring labels of their varied physical parameters. Partitioning the neural network parameters into *shared parameters* that capture the shared dynamics form and *adaptation parameters* that account for variations across system instances, a meta-trained iMODE can adapt to a new system instance by updating the (often low-dimensional) adaptation parameters for only several gradient descent steps, with scarce observations. After adaptation, the adaptation parameters correlate well with the unknown physical parameters of the systems, and therefore can be used to reveal knowledge of a family of dynamical systems.

From the other perspective, the "lens" of dynamical systems can also be applied to deep neural networks. The optimization of a deep neural network is a dynamic process where the optimizer navigates in the loss landscape [SBC16, SDJ22]. With the analysis of the optimization dynamics of deep neural networks, we propose Hessian-aware (HA-SGD), a novel gradient descent algorithm that searches for flat minima within the network's loss landscape by introducing perturbations to the optimization process. Through device-aware simulations, HA-SGD has demonstrated its capability to considerably improve the noise resiliency of analog neural networks, which is crucial to the deployment of large-scale analog neural networks to analog CIM platforms, thereby heralding a promising direction for orders of magnitude greater inference throughput and energy efficiency.

5.2 Future Work

In the future, the following open questions may be studied following this dissertation.

5.2.1 Physics-informed Deep Learning for Dynamics Modeling

The ESNN introduced in Chapter 3 focuses on the 2D reduced-order modeling of a very specific system, the Slinky. Concrete extensions to this work will include broadening the applicable scenarios of the method. One example is to generalize the reduced order modeling of Slinkies to 3D to capture the motion where the center line of the Slinky does not stay within one plane. Another example is to endow the neural model with the capability of capturing the dynamics of collision from data, which ESNN currently lacks.

More general extensions in the direction of physics-informed deep learning for modeling dynamical systems include

Data-driven learning of interpretable reduced-order state The reduced-order representation of Slinky in Chapter 3 is heuristically designed to ensure interpretability and to demonstrate the power of data-driven dynamics modeling. In more general cases of reduced-order modeling, manual design of the reduced-order representation of system states can be spared by data-driven learning-based methods. However, it is a challenge to obtain *disentangled*, *parsimonious*, and *interpretable* representation from data-driven approaches, especially deep learning. Such reduced-order representations are desirable in many applications, as they allow for more regularized modeling, easier control and diagnosis, and provide more insights.

General-purpose Equivariant neural networks The ESNN presented in Chapter 3 is designed specifically for the 2D representation of Slinky, but the incorporation of symmetry is valuable to the modeling of many systems. As mentioned in the introduction, there have been works of geometric deep learning [AGS21, BBC21] and equivariant neural networks [WGW18, TSK18, FWW21, SHW21] that aim to design general-purpose symmetry-aware neural network architectures. However, the task is still far from complete. Currently the trade-off between the generality and trainability of the symmetry-aware architectures limits the usage in many cases.

High-trainability neural ODE Although powerful and useful for the incorporation of

physics prior knowledge, we observe that the neural ODE framework can be unstable and inefficient to train in practice, mostly due to the loss that easily diverges during the "autoregressive" prediction following the neural ODE. Improving neural ODE in terms of its trainability is a field being actively explored [FJN20, KBJ20, LP21].

Uncertainty-aware and prior-knowledge-guided extrapolation In both Chapters 3 and 4, part of the reason that the neural ODE gets unstable to train is that the neural predictor for energy or force is not well regularized for unseen states. For example, if the elastic energy predictor of ESNN in Chapter 3 is not bounded below in the state space when the state goes to infinity in some direction, the resulted dynamics gets easily diverged because the energy can be infinitely lowered as the state diverges. It is desirable that the neural network is aware of the uncertainty of its prediction when there are not enough training data supports its predictions, and further extrapolates with caution following prior knowledge, e.g. the potential energy of a bounded system should go to infinity whenever the state gets out of the bounded region, while the "bounded region" may not be fixed but allows for data-driven learning.

5.2.2 Deep Learning for Automated Knowledge Discovery

We demonstrate in Chapter 4 that iMODE as a deep learning method can reveal physics knowledge of dynamical systems, which has become a popular research topic [KKL21, KPG22]. Recently deep learning has been used to discover from data the physics knowledge including conservation law [LT21, LMT22], symmetries [LT22], system coordinates [CLK19, CHR22], and other physical properties [ZWT18]. As the AI-assisted scientific research arises as a new paradigm, countless problems and opportunities remain to be explored.

5.2.3 Beyond First-order Optimization for DNNs Derived From Dynamics Analysis

A trend in optimization techniques for deep learning is to go beyond the first-order methods and leverage second-order information. As shown in Chapter 2, modification in dynamics can bring in second-order information, which can be further explored and exploited to facilitate second-order optimization of DNNs. Conversely, the estimation of the Hessian information in the proposed HA-SGD can be improved in terms of its accuracy and efficiency by leveraging techniques from second-order optimization algorithms.

REFERENCES

[AGS21] Kenneth Atz, Francesca Grisoni, and Gisbert Schneider. "Geometric Deep Learning on Molecular Representations." *Nature Machine Intelligence*, **3**(12):1023–1032, December 2021.

[BAV10] Miklós Bergou, Basile Audoly, Etienne Vouga, Max Wardetzky, and Eitan Grinspun. "Discrete Viscous Threads." *ACM Transactions on Graphics*, **29**(4):116:1–116:10, July 2010.

[BBC21] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges." *arXiv:2104.13478 [cs, stat]*, May 2021. Comment: 156 pages. Work in progress – comments welcome!

[BBS16] G. W. Burr, M. J. Brightsky, A. Sebastian, H. Cheng, J. Wu, S. Kim, N. E. Sosa, N. Papandreou, H. Lung, H. Pozidis, E. Eleftheriou, and C. H. Lam. "Recent Progress in Phase-Change Memory Technology." *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **6**(2):146–162, June 2016.

[BOC17] Peter Benner, Mario Ohlberger, Albert Cohen, and Karen Willcox. *Model Reduction and Approximation*. Computational Science & Engineering. Society for Industrial and Applied Mathematics, July 2017.

[BWR08] Miklós Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. "Discrete Elastic Rods." In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pp. 63:1–63:12, New York, NY, USA, 2008. ACM.

[CCS16] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. "Entropy-SGD: Biasing Gradient Descent Into Wide Valleys." *arXiv:1611.01838 [cs, stat]*, November 2016. Comment: ICLR '17.

[CGH20] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. "Lagrangian Neural Networks." *arXiv:2003.04630 [physics, stat]*, July 2020. Comment: 7 pages (+2 appendix). Published in ICLR 2020 Deep Differential Equations Workshop. Code at github.com/MilesCranmer/lagrangian_nns.

[CGV19] Alvin J. K. Chua, Chad R. Galley, and Michele Vallisneri. "Reduced-Order Modeling with Artificial Neurons for Gravitational-Wave Inference." *Physical Review Letters*, **122**(21):211101, May 2019.

[CHR22] Boyuan Chen, Kuang Huang, Sunand Raghupathi, Ishaan Chandratreya, Qiang Du, and Hod Lipson. "Automated Discovery of Fundamental Variables Hidden in Experimental Data." *Nature Computational Science*, **2**(7):433–442, July 2022.

[CKE17] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks." *IEEE Journal of Solid-State Circuits*, **52**(1):127–138, January 2017.

[CLK19] Kathleen Champion, Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. "Data-Driven Discovery of Coordinates and Governing Equations." *Proceedings of the National Academy of Sciences*, **116**(45):22445–22451, November 2019.

[CRB18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural Ordinary Differential Equations." In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[CWK19] Taco S. Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. "Gauge Equivariant Convolutional Networks and the Icosahedral CNN." *arXiv:1902.04615 [cs, stat]*, May 2019. Comment: Proceedings of the International Conference on Machine Learning (ICML), 2019.

[DMJ22] Shaan Desai, Marios Mattheakis, Hayden Joy, Pavlos Protopapas, and Stephen J. Roberts. "One-Shot Transfer Learning of Physics-Informed Neural Networks." In *ICML 2022 2nd AI for Science Workshop*, July 2022.

[DPB17] Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. "Sharp Minima Can Generalize For Deep Nets." *arXiv:1703.04933 [cs]*, March 2017. Comment: 8.5 pages of main content, 2.5 of bibliography and 1 page of appendix.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In *Proceedings of the 34th International Conference on Machine Learning*, pp. 1126–1135. PMLR, July 2017.

[FJN20] Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman. "How to Train Your Neural ODE: The World of Jacobian and Kinetic Regularization." In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3154–3164. PMLR, November 2020.

[FMD19] Lawson Fulton, Vismay Modi, David Duvenaud, David I. W. Levin, and Alec Jacobson. "Latent-Space Dynamics for Reduced Deformable Simulation." *Computer Graphics Forum*, **38**(2):379–391, 2019.

[FRK19] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. "Online Meta-Learning." In *Proceedings of the 36th International Conference on Machine Learning*, pp. 1920–1930. PMLR, May 2019.

[FWW21] Marc Finzi, Max Welling, and Andrew Gordon Wilson. "A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups." *arXiv:2104.09459 [cs, math, stat]*, April

2021. Comment: Library: https://github.com/mfinzi/equivariant-MLP, Documentation: https://emlp.readthedocs.io/en/latest/, Examples: https://colab.research.google.com/github/mfinzi/equivariant-MLP/blob/master/docs/notebooks/colabs/all.ipynb.

[FWY20] Romain Fournier, Lei Wang, Oleg V. Yazyev, and QuanSheng Wu. "Artificial Neural Network Approach to the Analytic Continuation Problem." *Physical Review Letters*, **124**(5):056401, February 2020.

[GBB17] X. Guo, F. M. Bayat, M. Bavandpour, M. Klachko, M. R. Mahmoodi, M. Prezioso, K. K. Likharev, and D. B. Strukov. "Fast, Energy-Efficient, Robust, and Reproducible Mixed-Signal Neuromorphic Classifier Based on Embedded NOR Flash Memory Technology." In *2017 IEEE International Electron Devices Meeting (IEDM)*, pp. 6.5.1–6.5.4, December 2017.

[GDY19] Sam Greydanus, Misko Dzamba, and Jason Yosinski. "Hamiltonian Neural Networks." *arXiv:1906.01563 [cs]*, September 2019. Comment: Conference paper at NeurIPS 2019. Main paper has 8 pages and 5 figures.

[GHS16] Erik Gawehn, Jan A. Hiss, and Gisbert Schneider. "Deep Learning in Drug Discovery." *Molecular Informatics*, **35**(1):3–14, 2016.

[GS22] Mario Geiger and Tess Smidt. "E3nn: Euclidean Neural Networks.", July 2022. Comment: draft.

[GSR17] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural Message Passing for Quantum Chemistry." In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272, Sydney, NSW, Australia, August 2017. JMLR.org.

[GWI19] X. Gu, Z. Wan, and S. S. Iyer. "Charge-Trap Transistors for CMOS-Only Analog Memory." *IEEE Transactions on Electron Devices*, pp. 1–5, 2019.

[HBM14] Douglas P. Holmes, Andy D. Borum, Billy F. Moore, Raymond H. Plaut, and David A. Dillard. "Equilibria and Instabilities of a Slinky: Discrete Model." *International Journal of Non-Linear Mechanics*, **65**:236–244, October 2014.

[HGP19] W. Haensch, T. Gokmen, and R. Puri. "The Next Generation of Deep Learning Hardware: Analog Computing." *Proceedings of the IEEE*, **107**(1):108–122, January 2019.

[HHM22] Jiaqi Han, Wenbing Huang, Hengbo Ma, Jiachen Li, Joshua B. Tenenbaum, and Chuang Gan. "Learning Physical Dynamics with Subequivariant Graph Neural Networks.", October 2022. Comment: Accepted by NeurIPS 2022.

[HHR21] Wenbing Huang, Jiaqi Han, Yu Rong, Tingyang Xu, Fuchun Sun, and Junzhou Huang. "Equivariant Graph Mechanics Networks with Constraints." In *International Conference on Learning Representations*, September 2021.

[HLM17] Yashar D. Hezaveh, Laurence Perreault Levasseur, and Philip J. Marshall. "Fast Automated Analysis of Strong Gravitational Lenses with Convolutional Neural Networks." *Nature*, **548**(7669):555–557, August 2017.

[HLv16] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks." *arXiv:1608.06993 [cs]*, August 2016. Comment: CVPR 2017.

[HM13] Jennifer Hasler and Harry Bo Marr. "Finding a Roadmap to Achieve Large Neuromorphic Hardware Systems." *Frontiers in Neuroscience*, **7**, 2013.

[HM17] David Hartman and Lalit K. Mestha. "A Deep Learning Framework for Model Reduction of Dynamical Systems." In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 1917–1922, August 2017.

[HMC21] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S. Sukhatme. "NeuralSim: Augmenting Differentiable Simulators with Neural Networks." *arXiv:2011.04217 [cs]*, May 2021. Comment: Accepted at IEEE International Conference on Robotics and Automation (ICRA) 2021.

[HRM21] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. "A Physics-Informed Deep Learning Framework for Inversion and Surrogate Modeling in Solid Mechanics." *Computer Methods in Applied Mechanics and Engineering*, **379**:113741, June 2021.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Flat Minima." *Neural Computation*, **9**(1):1–42, January 1997.

[HZR16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition." In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[IMW20] Raban Iten, Tony Metger, Henrik Wilming, Lídia del Rio, and Renato Renner. "Discovering Physical Concepts with Neural Networks." *Physical Review Letters*, **124**(1):010508, January 2020.

[JDJ14] Mohammad K. Jawed, Fang Da, Jungseock Joo, Eitan Grinspun, and Pedro M. Reis. "Coiling of Elastic Rods on Rigid Substrates." *Proceedings of the National Academy of Sciences*, **111**(41):14663–14668, October 2014.

[JGW21] José Jiménez-Luna, Francesca Grisoni, Nils Weskamp, and Gisbert Schneider. "Artificial Intelligence in Drug Discovery: Recent Advances and Future Perspectives." *Expert Opinion on Drug Discovery*, **16**(9):949–959, September 2021.

[JNO18] M. Khalid Jawed, Alyssa Novelia, and Oliver M. O'Reilly. *A Primer on the Kinematics of Discrete Elastic Rods*. SpringerBriefs in Applied Sciences and Technology. Springer International Publishing, Cham, 2018.

[JYP17] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. "In-Datacenter Performance Analysis of a Tensor Processing Unit." In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pp. 1–12, New York, NY, USA, 2017. ACM.

[KB14] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." *arXiv:1412.6980 [cs]*, December 2014. Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[KBJ20] Jacob Kelly, Jesse Bettencourt, Matthew James Johnson, and David Duvenaud. "Learning Differential Equations That Are Easy to Solve." *arXiv:2007.04504 [cs, stat]*, October 2020.

[KJM10] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. "Efficient Yarn-Based Cloth with Adaptive Contact Linearization." *ACM Transactions on Graphics*, **29**(4):105:1–105:10, July 2010.

[KKL21] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. "Physics-Informed Machine Learning." *Nature Reviews Physics*, **3**(6):422–440, June 2021.

[KMN16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." *arXiv:1609.04836 [cs, math]*, September 2016. Comment: Accepted as a conference paper at ICLR 2017.

[KMT19] Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. "Energy Flow Networks: Deep Sets for Particle Jets." *Journal of High Energy Physics*, **2019**(1):121, January 2019. Comment: 31+16 pages, 21 figures, 5 tables; v2: updated to match JHEP version; code available at https://energyflow.network.

[KPG22] Mario Krenn, Robert Pollice, Si Yue Guo, Matteo Aldeghi, Alba Cervera-Lierta, Pascal Friederich, Gabriel dos Passos Gomes, Florian Häse, Adrian Jinich, AkshatKumar Nigam, Zhenpeng Yao, and Alán Aspuru-Guzik. "On Scientific Understanding with Artificial Intelligence." *Nature Reviews Physics*, **4**(12):761–769, December 2022.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature*, **521**(7553):436–444, May 2015.

[LBL18] Can Li, Daniel Belkin, Yunning Li, Peng Yan, Miao Hu, Ning Ge, Hao Jiang, Eric Montgomery, Peng Lin, Zhongrui Wang, Wenhao Song, John Paul Strachan, Mark Barnell, Qing Wu, R. Stanley Williams, J. Joshua Yang, and Qiangfei Xia. "Efficient and Self-Adaptive in-Situ Learning in Multilayer Memristor Neural Networks." *Nature Communications*, **9**(1):2385, June 2018.

[LFS20] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. "Incremental Potential Contact: Intersection-and Inversion-Free, Large-Deformation Dynamics." *ACM Transactions on Graphics*, **39**(4):49:49:1–49:49:20, August 2020.

[LKA20] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Fourier Neural Operator for Parametric Partial Differential Equations." In *International Conference on Learning Representations*, September 2020.

[LKB18] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. "Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics." *Nature Communications*, **9**(1):4950, November 2018.

[LKK18] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H. Yoo. "UNPU: A 50.6TOPS/W Unified Deep Neural Network Accelerator with 1b-to-16b Fully-Variable Weight Bit-Precision." In *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, pp. 218–220, February 2018.

[LMT22] Ziming Liu, Varun Madhavan, and Max Tegmark. "AI Poincar\'{e} 2.0: Machine Learning Conservation Laws from Differential Equations." *arXiv:2203.12610 [astro-ph, physics:nlin, physics:physics]*, March 2022. Comment: 17 pages, 10 figures.

[LNB18] Jean-Christophe Loiseau, Bernd R. Noack, and Steven L. Brunton. "Sparse Reduced-Order Modelling: Sensor-Based Dynamics to Full-State Estimation." *Journal of Fluid Mechanics*, **844**:459–490, June 2018.

[LP21] Kookjin Lee and Eric J. Parish. "Parameterized Neural Ordinary Differential Equations: Applications to Computational Physics Problems." *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **477**(2253):20210162, September 2021.

[LRP18] Michael Lutter, Christian Ritter, and Jan Peters. "Deep Lagrangian Networks: Using Physics as Model Prior for Deep Learning." In *International Conference on Learning Representations*, September 2018.

[LT21] Ziming Liu and Max Tegmark. "Machine Learning Conservation Laws from Trajectories." *Physical Review Letters*, **126**(18):180604, May 2021.

[LT22] Ziming Liu and Max Tegmark. "Machine Learning Hidden Symmetries." *Physical Review Letters*, **128**(18):180201, May 2022.

[LWR23] Qiaofeng Li, Tianyi Wang, Vwani Roychowdhury, and M. Khalid Jawed. "Rapidly Encoding Generalizable Dynamics in a Euclidean Symmetric Neural Network." *Extreme Mechanics Letters*, **58**:101925, January 2023.

[LZK21] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. "Physics-Informed Neural Operator for Learning Partial Differential Equations." *arXiv:2111.03794 [cs, math]*, November 2021.

[MML20] Romit Maulik, Arvind Mohan, Bethany Lusch, Sandeep Madireddy, Prasanna Balaprakash, and Daniel Livescu. "Time-Series Learning of Latent-Space Dynamics for Reduced-Order Model Closure." *Physica D: Nonlinear Phenomena*, **405**:132368, April 2020.

[MQC18] W. Ma, M. Qin, W. H. Choi, P. Chiu, and M. Lueker-Boden. "Improving Noise Tolerance of Hardware Accelerated Artificial Neural Networks." In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 797–801, December 2018.

[NAS18] Alex Nichol, Joshua Achiam, and John Schulman. "On First-Order Meta-Learning Algorithms." *arXiv:1803.02999 [cs]*, October 2018.

[NG20] Nirmal J. Nair and Andres Goza. "Leveraging Reduced-Order Models for State Estimation Using Deep Learning." *Journal of Fluid Mechanics*, **897**:R1, August 2020.

[PBJ22] Junyoung Park, Federico Berto, Arec Jamgochian, Mykel J. Kochenderfer, and Jinkyoo Park. "Meta-SysId: A Meta-Learning Approach for Simultaneous Identification and Prediction.", June 2022. Comment: 9 pages, 8 figures.

[PKI19] J. Panetta, M. Konaković-Luković, F. Isvoranu, E. Bouleau, and M. Pauly. "X-Shells: A New Class of Deployable Beam Structures." *ACM Transactions on Graphics*, **38**(4):83:1–83:15, July 2019.

[QG20] Huilin Qu and Loukas Gouskos. "ParticleNet: Jet Tagging via Particle Clouds." *Physical Review D*, **101**(5):056019, March 2020. Comment: 11 pages, 4 figures; v3: updated to match the version published in PRD; Code available at https://github.com/hqucms/ParticleNet.

[RBP17] Samuel H. Rudy, Steven L. Brunton, Joshua L. Proctor, and J. Nathan Kutz. "Data-Driven Discovery of Partial Differential Equations." *Science Advances*, **3**(4):e1602614, April 2017.

[RCD19] Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. "Latent ODEs for Irregularly-Sampled Time Series." *arXiv:1907.03907 [cs, stat]*, July 2019.

[RFK19] Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. "Meta-Learning with Implicit Gradients." In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[RGP18] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul N. Whatmough, Sae Kyu Lee, Niamh Mulholland, David M. Brooks, and Gu-Yeon Wei. "Ares: A Framework for Quantifying the Resilience of Deep Neural Networks." *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[RPK19] M. Raissi, P. Perdikaris, and G. E. Karniadakis. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations." *Journal of Computational Physics*, **378**:686–707, February 2019.

[RRB19] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. "Rapid Learning or Feature Reuse? Towards Understanding the Effectiveness of MAML." In *International Conference on Learning Representations*, September 2019.

[SBC16] Weijie Su, Stephen Boyd, and Emmanuel J. Candès. "A Differential Equation for Modeling Nesterov's Accelerated Gradient Method: Theory and Insights." *The Journal of Machine Learning Research*, **17**(1):5312–5354, January 2016.

[SDJ22] Bin Shi, Simon S. Du, Michael I. Jordan, and Weijie J. Su. "Understanding the Acceleration Phenomenon via High-Resolution Differential Equations." *Mathematical Programming*, **195**(1-2):79–148, September 2022.

[SHW21] Victor Garcia Satorras, Emiel Hoogeboom, and Max Welling. "E(n) Equivariant Graph Neural Networks." *arXiv:2102.09844 [cs, stat]*, May 2021.

[SLY18] Yuanyuan Shi, Xianhu Liang, Bin Yuan, Victoria Chen, Haitong Li, Fei Hui, Zhouchangwan Yu, Fang Yuan, Eric Pop, H.-S. Philip Wong, and Mario Lanza. "Electronic Synapses Made of Layered Two-Dimensional Materials." *Nature Electronics*, **1**(8):458–465, August 2018.

[SPH16] Abhronil Sengupta, Maryam Parsa, Bing Han, and Kaushik Roy. "Probabilistic Deep Spiking Neural Systems Enabled by Magnetic Tunnel Junction." *IEEE Transactions on Electron Devices*, **63**(7):2963–2970, July 2016.

[TSK18] Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. "Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds." *arXiv:1802.08219 [cs]*, May 2018. Comment: changes for NIPS submission.

[Wan20] Zhe Wan. *Scalable and Analog Neuromorphic Computing Systems*. PhD thesis, UCLA, 2020.

[WGW18] Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco Cohen. "3D Steerable CNNs: Learning Rotationally Equivariant Features in Volumetric Data." In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 10402–10413, Red Hook, NY, USA, December 2018. Curran Associates Inc.

[Wil17] R. S. Williams. "What's Next? [The End of Moore's Law]." *Computing in Science Engineering*, **19**(2):7–13, March 2017.

[WZL21] Haoxiang Wang, Han Zhao, and Bo Li. "Bridging Multi-Task Learning and Meta-Learning: Towards Efficient Training and Effective Adaptation." In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10991–11002. PMLR, July 2021.

[XLS19] Zhenjia Xu, Zhijian Liu, Chen Sun, Kevin Murphy, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. "Unsupervised Discovery of Parts, Structure, and Dynamics." In *International Conference on Learning Representations*, December 2019.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks." *arXiv:1605.07146 [cs]*, May 2016.

[ZSZ18] Pengfei Zhang, Huitao Shen, and Hui Zhai. "Machine Learning Topological Invariants with Neural Networks." *Physical Review Letters*, **120**(6):066401, February 2018.

[ZWT18] David Y. Zheng, Jiajun Wu, and Joshua B. Tenenbaum. "Unsupervised Learning of Latent Physical Properties Using Perception-Prediction Networks." In *Conference on Uncertainty in Artificial Intelligence (UAI)*. Association For Uncertainty in Artificial Intelligence (AUAI), June 2018. Comment: UAI 2018 (oral).

[ZZP18] Xin Zheng, Ryan Zarcone, Dylan Paiton, Joon Sohn, Weier Wan, Bruno Olshausen, and H. S. Philip Wong. "Error-Resilient Analog Image Storage and Compression with Analog-Valued RRAM Arrays: An Adaptive Joint Source-Channel Coding Approach." In *2018 IEEE International Electron Devices Meeting (IEDM)*, pp. 3.5.1–3.5.4, December 2018.