

A study on quantifying effective training of DLDMD

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Applied Mathematics

with a Concentration in

Dynamical Systems

by

Joseph Aaron Gene Diaz

August 2, 2022

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Joseph Aaron Gene Diaz:

A study on quantifying effective training of DLDMD

Christopher Curtis, Chair
Department of Mathematics and Statistics

Jérôme Gilles
Department of Mathematics and Statistics

Saeed Manshadi
Department of Electrical Engineering

Approval Date

Copyright © 2022
by
Joseph Aaron Gene Diaz

DEDICATION

Everyone who's helped me get this far: You know who you are.

ABSTRACT OF THE THESIS

A study on quantifying effective training of DLDMD
by

Joseph Aaron Gene Diaz

Master of Science in Applied Mathematics with a Concentration in Dynamical Systems
San Diego State University, 2022

Machine Learning is a useful tool for many problems in modern applied mathematics, especially given the availability of data and recent data-driven methods of analysis. As with any such method, quantifying the effectiveness of training is a meaningful question to ask when standard metrics do not give a concrete answer.

TABLE OF CONTENTS

	PAGE
ABSTRACT	v
LIST OF TABLES	vii
LIST OF FIGURES	viii
CHAPTER	
1 INTRODUCTION	1
1.1 What is Dynamic Mode Decomposition?	1
1.2 What are Neural Networks?	3
1.3 DLDMD and Its Limitations	5
1.4 Network Architecture of DLDMD.....	6
1.5 Statement of Dilemma	7
2 KULLBACK-LEIBLER DIVERGENCE	9
2.1 Basic Definitions	9
2.2 How to build distributions (Non-parametric statistics).....	11
2.3 Measuring entropy flow	12
2.4 Implementation Details	13
3 RESULTS	17
3.1 The Duffing Oscillator	17
3.2 The Van der Pol Oscillator	22
4 DISCUSSION	32
4.1 The Good	32
4.2 Optimal Parameters.....	33
4.3 Non-unitary densities	33
4.4 Statistical issues with edge layers	34
4.5 Future work.....	34
BIBLIOGRAPHY	36

LIST OF TABLES

	PAGE
3.1 Hyperparamters of the DLDMD algorithm held constant for each training run on Duffing.	18
3.2 ReLU activation function vs ELU activation function. Unlike ReLU, ELU is negative for $x < 0$. For some models, ELU can be advantageous if ReLU causes many of the neuron to “die” during training which means the weights associated with that neuron become 0, and those neurons do not contribute anything to output of the machine. Other activation functions can assuage this problem, but ELU can be made arbitrarily close to ReLU and so is quite popular.	19

LIST OF FIGURES

	PAGE
1.1 Example of DLDMD newtork with $N_S = 2$, $N_O = 4$, and $N_L = 3$ where every hidden layer has 16 neurons. This representation is meaningful in that it gives the name of each layer as they appear from left to right, but it is missing some key network features like the activation functions.....	7
2.1 Entropy and density function change with respect to σ^2 . The base of the logarithm shown here is e , but the base itself is irrelevant due to the shared properties of all logarithms: that they increase monotonically and unbounded. As the variance increases we see higher entropy in the left plot and a flattening of the probability density in the right plot.	10
2.2 Kullback-Leibler Divergence example with two normal distributions. As expected, the divergence decreases to 0 as $\sigma^2 \rightarrow 1$ from either direction.	11
2.3 Example of a density function found using KDE, its shape coincides with that of the histogram generated from the same samples.	13
2.4 Example of linear fitting of divergence data for one of the models we examine in Chapter 3.	16
3.1 Duffing phase space.	18
3.2 Duffing average slopes of linear fits with standard deviations. This plot illustrates what we want to find. Each latent dimension gives us a different behavior and we can readily see how the information must be changing during training.	20
3.3 Duffing loss plots, smoothed to accentuate trends.	21
3.4 Duffing Average bandwidth.	21
3.5 Duffing slope of linear fits by layer.	23
3.6 Duffing linear fits by layer.	24
3.7 Duffing DLDMD results by latent dimension.	25
3.8 Van der Pol phase space.	26
3.9 Van der Pol average slopes with standard deviations.	27
3.10 Van der Pol loss plots, smoothed to accentuate trend.	27
3.11 Van der Pol Average bandwidth.	28
3.12 Van der Pol slope of linear fits by layer.	29

3.13	Van der Pol linear fits by layer.....	30
3.14	Van der Pol DLDMD results by latent dimension.....	31

CHAPTER 1

INTRODUCTION

In the study of dynamical systems a central problem is how to derive models from measured data to facilitate the prediction of future states. Many approaches and techniques exist in the literature, from deriving sets of governing equations via application of the simple physical principles to the statistically meaningful Principal Component Analysis and other modal decompositions. The main goal of many of these methods is to come up with a generalized framework so that the dynamics can be extracted from the data for the sake of control and prediction. While there are genuine advantages to the more concrete and scientifically-sound methods of deriving models from first principles, it is often very difficult if not outright untenable.

As such, data-driven methods have rapidly become very useful approaches for coming up with rough and ready models for prediction. A collection of such methods exploit the Koopman operator to find numerical approximations of underlying dynamics to forecast future states. These are called Dynamic Mode Decompositions and they fit nicely within the broad family of modal decompositions. In the coming sections, we will introduce a specific strain of Dynamic Mode Decomposition and discuss improvements made to it using Machine Learning (ML). The ultimate goal of this thesis is to show that we have come up with a method for quantifying the effectiveness of the training that the affiliated network undergoes with a few computed indicators.

1.1 What is Dynamic Mode Decomposition?

As it says on the tin, we seek a predictive model for a time series $\{\mathbf{y}_j\}_{j=1}^{N^T+1}$, which are the measurements of a uniformly sampled unknown dynamical system of the form

$$\frac{d\mathbf{y}}{dt} = f(\mathbf{y}(t)), \quad \mathbf{y}(0) = \mathbf{x} \in \mathcal{M} \subseteq \mathbb{R}^{N_s} \quad (1.1)$$

As established in the introduction, there is an old and venerable literature dedicated to deriving systems using classical methods. What this literature will admit is that such a process is not trivial and even difficult or impossible to do in practice; particularly for nonlinear phenomena worth investigating. We want something that is more easily generalized and algorithmic.

One method that can be leveraged for this is via the Koopman Operator \mathcal{K} . If we denote $\varphi(t; \mathbf{x}) = \mathbf{y}(t)$ to be the flow map affiliated with the initial condition \mathbf{x} and

denote $g : \mathcal{M} \rightarrow \mathbb{C}$ to be a square integrable observable of the system then, according to [8], there exists a linear representation of the flow map given by \mathcal{K} ;

$$\mathcal{K}^t g(\mathbf{x}) = g(\boldsymbol{\varphi}(t; \mathbf{x})), \quad (1.2)$$

which means that \mathcal{K} is a linear time-advancement operator of the dynamics. This might seem like it trivializes the problem, given that we now have a linear system, but it does not. The Koopman operator is an infinite-dimensional operator, so we have traded a potentially nonlinear problem for an infinite-dimensional linear one.

In order to actually use this new representation, it suffices to find the eigenvalues $\{\lambda_\ell\}$, and their affiliated eigenfunctions $\{\boldsymbol{\phi}_\ell\}$, of the Koopman Operator such that

$$\mathcal{K}^t \boldsymbol{\phi}_\ell = \exp(t\lambda_\ell) \boldsymbol{\phi}_\ell \implies g(\mathbf{x}) = \sum_{\ell \in \mathbb{N}} a_\ell \boldsymbol{\phi}_\ell(\mathbf{x}), \quad (1.3)$$

where we can essentially construct a modal decomposition of g . From here advancing the dynamics to time t is equivalent to writing

$$\mathcal{K}^t g(\mathbf{x}) = \sum_{\ell \in \mathbb{N}} a_\ell \exp(t\lambda_\ell) \boldsymbol{\phi}_\ell(\mathbf{x}) \quad (1.4)$$

The most useful property of this framing is that we have a global linearization of the flow, with a major caveat: finding these eigenvalues and eigenfunctions is usually impossible. This led to the development of the already mentioned and much lauded Dynamic Mode Decomposition (DMD) [15] and its extensions, which seeks to numerically approximate a finite number of these modes. We will focus on the particular extension that [1] implements: Extended Dynamic Mode Decomposition (EDMD) [20]. With EDMD, we suppose that

$$g(\mathbf{x}) = \sum_{\ell=1}^{N_O} a_\ell \boldsymbol{\psi}_\ell(\mathbf{x}) \quad (1.5)$$

which is to say that the observable g exists in a N_O -dimensional subspace of $L^2(\mathcal{M})$ with basis functions $\{\boldsymbol{\psi}_\ell\}_{\ell=1}^{N_O}$, and applying the Koopman operator implies that

$$\mathcal{K}^{\delta t} g(\mathbf{x}) = \sum_{\ell=1}^{N_O} a_\ell \exp(\delta t \lambda_\ell) \boldsymbol{\psi}_\ell(\mathbf{x}) \quad (1.6)$$

$$= \sum_{\ell=1}^{N_O} \boldsymbol{\psi}_\ell(\mathbf{x}) (\mathbf{K}_O^T \mathbf{a})_\ell + r(\mathbf{x}; \mathbf{K}_O) \quad (1.7)$$

for discrete time step δt . \mathbf{K}_O is the $N_O \times N_O$ matrix that minimizes

$$\mathbf{K}_O = \underset{K}{\operatorname{argmin}} \| \boldsymbol{\Psi}_+ - K \boldsymbol{\Psi}_- \|_F^2 \quad (1.8)$$

and $r(\mathbf{x}, \mathbf{K}_O)$ is a residual that represents the total error due to DMD. If the ansatz that g lives in a finite dimensional space holds, then r is identically 0. We define Ψ_{\pm} to be

$$\Psi_- = (\Psi_1 \ \Psi_2 \ \dots \ \Psi_{N_T}), \quad \Psi_+ = (\Psi_2 \ \Psi_3 \ \dots \ \Psi_{N_T+1}) \quad (1.9)$$

where $\{\Psi_j\}$ is an observable of the time series of interest $\{\mathbf{y}_j\}$. What the expression for \mathbf{K}_O tells us is that, we are trying to find a one-step mapping from each data point to the next. Practically speaking, \mathbf{K}_O is found using a singular value decomposition (SVD), with which we can write

$$\Psi_- = \mathbf{U}\Sigma\mathbf{W}^\dagger \implies \mathbf{K}_O = \Psi_+\mathbf{W}\Sigma^{-P}\mathbf{U}^\dagger \quad (1.10)$$

where $-P$ denotes the Moore-Penrose pseudo-inverse and \dagger denotes the conjugate transpose. This gives us an expression for r in terms of the observables Ψ :

$$E_r(\mathbf{K}_O) = \|\Psi_+(\mathbf{I} - \mathbf{W}\mathbf{W}^\dagger)\|_F \quad (1.11)$$

Finding the eigenvalues, eigenfunctions, and Koopman modes comes down to an eigen-decomposition of \mathbf{K}_O ,

$$\mathbf{K}_O = \mathbf{V}\mathbf{T}\mathbf{V}^{-1}, \quad (1.12)$$

with $\lambda_\ell = \ln((\mathbf{T})_{\ell\ell})/\delta t$, from which the dynamics can be approximated as

$$y(t; \mathbf{x}) \approx \mathbf{K}_m \exp(t\Lambda) \mathbf{V}^{-1} \Psi(\mathbf{x}) \quad (1.13)$$

where Ψ is the representation of the initial condition in terms of the observables, \mathbf{K}_m is the $N_S \times N_O$ matrix whose columns are the Koopman modes $\{\mathbf{k}_\ell\}_{\ell=1}^{N_O}$ which solve the initial value problem

$$\mathbf{x} = \sum_{\ell=1}^{N_O} \mathbf{k}_\ell \phi(\mathbf{x}), \quad (1.14)$$

and Λ is the diagonal matrix whose elements are $\lambda_\ell = (\Lambda)_{\ell\ell}$.

1.2 What are Neural Networks?

Before we move on, a brief digression on Neural Networks (NNs) is appropriate. Ever since scientists discovered the relatively simple interaction between neurons and axons in the human brain, they have been enamored with the ability to create a learning computer with the similar ability to become more adept at a task with training and practice. In the same way that art imitates life, the most straight-forward attempts have been to construct artificial neural networks that pantomime the biological ones that we carry around in our heads; and while these pale imitations have not been

developed to rival the human brain, they have led to some major achievements in automation. In much the same way that a person is “trained” to perform a task by repeating it with feedback and adjusting their performance as they go, an artificial NN uses data, a loss function, and an optimization algorithm to change the state of the NN to one which can better accomplish the task. The loss function and optimizer are to the feedback and behavioral adjustment what the data is to the experience.

The oldest known example of NNs were the perceptrons of the 1950s, which could perform binary classifications of linearly separable data. Multilayer versions of this simple architecture allowed for classification of larger set of classes, but the researchers of the time arrived at the limits of their computing environments rather quickly and a brief dark age in the study and application of neural networks ensued. Interest was renewed in the 1980s when many of the hardware limitations of previous decades were ameliorated and new algorithms for optimizing had been pioneered [14].

In modern mathematical study and application, the question of function representation is an ever-present one; specifically with respect to approximating quantities that can be given via series. While truncations and use of the likes of Fourier series are useful and effective in some narrow applications, one means of representation that has shown much promise in recent years due to the greater availability of computing power is through the use of NNs. In 1989, [7] showed that, with some appropriate parameters, a NN can be used to approximate any continuous function from one vector space to another to any arbitrary precision required. The means of how this approximation is generated from measured data is one that statisticians have studied for decades. How can one produce a model that respects that data used to generate it without underfitting or overfitting?

With respect to function approximation, we can consider ML through NNs as a tool of regression [5, 19]. Say we have a set of training data $d_{tr} = \{(x_j, y_j)\}_{j=1}^{N_{tr}}$ and we believe that there is some function $f(x)$ such that

$$y_j = f(x_j) + \varepsilon_j, \quad (1.15)$$

where ε_j is assumed to be a sample drawn from $\mathcal{N}(0, \bar{\sigma}^2)$. Generally f is not known; so we must, using the training data, build an estimator $f_e(x; d_{tr})$ and measure the performance of the estimator by examining the quantity

$$\mathbb{E}_{d_{tr}} [(f(x) + \varepsilon - f_e(x; d_{tr}))^2] = \text{bias}_{d_{tr}}^2 + \mathbb{V}(f_e) + \bar{\sigma}^2 \quad (1.16)$$

where

$$\text{bias}_{d_{tr}} = \mathbb{E}_{d_{tr}} [f_e] - f(x) \quad (1.17)$$

and

$$\mathbb{V}(f_e) = \mathbb{E}_{d_{tr}} [(\mathbb{E}_{d_{tr}} [f_e] - f_e(x; d_{tr}))^2] \quad (1.18)$$

With this framing, ML methods can be cast as attempts to find an estimator relative to a given training set that balances the competing dilemmas of being too biased or allowing too much variance. From the point of view of non-parametric statistics, we would need to utilize other techniques [6] to determine the bias and variance of any given estimator. For our purposes, we will instead consider how metrics from information theory can be used to assess the change in information as data is fed through a NN and a model is trained.

1.3 DLDMD and Its Limitations

The key innovation of [1] is to use a neural network to come up with the collection of observables on $\{\mathbf{y}_j\}$ that allow for the best prediction of future system states, we call this method Deep Learning Enhanced DMD (DLDMD). This is implemented by defining an encoder $\mathcal{E} : \mathbb{R}^{N_S} \rightarrow \mathbb{R}^{N_O}$ and decoder $\mathcal{D} : \mathbb{R}^{N_O} \rightarrow \mathbb{R}^{N_S}$ composed of dense layers such that

$$(\mathcal{D} \circ \mathcal{E})(\mathbf{x}) = \mathbf{x} \quad (1.19)$$

We choose $N_O \geq N_S$ and an appropriate loss function so that \mathcal{E} and \mathcal{D} give a richer space of observables, called the latent space, for EDMD to use when advancing the dynamics. The implementation of NNs for this purpose requires a method of tuning to allow \mathcal{E} and \mathcal{D} to learn the best representations possible. As such, a loss function that correctly identifies and prioritizes the desired properties is a necessary condition for the DLDMD to function as needed. A natural choice considering these constraints is given by

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{recon}} + \alpha_2 \mathcal{L}_{\text{dmd}} + \alpha_3 \mathcal{L}_{\text{pred}} + \alpha_4 \|\mathbf{W}_g\|_2 \quad (1.20)$$

where

$$\mathcal{L}_{\text{recon}} = \frac{1}{N_T + 1} \sum_{j=1}^{N_T+1} \|\mathbf{y}_j - (\mathcal{D} \circ \mathcal{E})(\mathbf{y}_j)\|_2, \quad (1.21)$$

$$\mathcal{L}_{\text{dmd}} = E_r(\mathbf{K}_O), \quad (1.22)$$

$$\mathcal{L}_{\text{pred}} = \frac{1}{N_T} \sum_{j=1}^{N_T} \|\mathbf{y}_{j+1} - \mathcal{D}(V T^j V^{-1} \mathcal{E}(\mathbf{x}))\|_2, \quad (1.23)$$

Recall that N_T is the number of measurements in our time-series data after the first. Each component guides the machine to a particular outcome:

1. $\mathcal{L}_{\text{recon}}$ is the mean-squared error (MSE) of each time step with respect to the reconstruction from the composition of \mathcal{E} and \mathcal{D} . This component ensures that, under training, the network effectively acts as a near identity transformation for data that is fed into it. This quality allows the DMD advanced trajectories to be recovered from the higher dimensional latent space back to the original dimension of the data.
2. \mathcal{L}_{DMD} is the error associated with DMD. Consequently, this component is the one that is most responsible for finding the optimal set of observables for DMD. \mathcal{E} immerses the data into a higher dimensional latent space, in effect acting as our set of observables; minimizing this gives us greater flexibility in the latent space.
3. $\mathcal{L}_{\text{pred}}$ is the MSE for each forward time prediction due to DMD and immersion/submersion due to \mathcal{E}/\mathcal{D} . In addition to balancing the last conditions, this condition ensures that the DMD step in the latent dimension is consistent with the next time-step from the time series after encoding, advancing, and decoding.
4. \mathbf{W}_g is the vectorized quantity that represents all weights in both \mathcal{E} and \mathcal{D} . This is really only a regularization condition to keep the coefficients of the weight matrices from blowing up in value as the model trains, which can be a concern for ML models.
5. α_1 , α_2 , α_3 , and α_4 are four positive constants that allow us to assign a weighting to each component of the loss. This allows the loss function to be dynamically weighted to prioritize some conditions over others. In [1], $\alpha_1 = \alpha_2 = \alpha_3 = 1$ and $\alpha_4 < 10^{-10}$.

The marriage of EDMD with dense NNs and some meaningful choices of loss function as an innovation was made by [1, 3, 11], and has proved to be quite a robust method for learning dynamics from data. In researching this thesis, many attempts were made to test the limits of DLDMD. [1] was primarily interested in recreating the phase space behavior of a handful of well known nonlinear oscillators such as the nonlinear harmonic oscillator, the Duffing oscillator, and the Van der Pol oscillator. Additionally, chaotic systems like Lorenz 63 and the Rössler system were examined but chaos proved to be quite a challenge for the algorithm to overcome. A study conducted by the author for a class project into the addition of Gaussian noise led to the discovery of the machine being quite robust to noisy data due to the characteristics of its loss function and additional noise mitigation was accomplished by implementing convolutional layers instead of traditional dense layers.

1.4 Network Architecture of DLDMD

This discussion leads naturally to the architecture of the network itself. The composition of the decoder with the encoder $\mathcal{D} \circ \mathcal{E}$ gives an Autoencoder [9] that

actually learns the optimal embeddings and imbeddings that we use in the DMD step of DLDMD. The encoder and decoder are sequential networks that accepts data at the input layer, runs them through the hidden layers, and yields them at the output layer. For convenience we label the input layers Enc in and Dec in, the hidden layers Enc l and Dec l for the hidden layers, and the output layers Enc out and Dec out for the encoder and decoder, respectively. The number of hidden layers N_L is a model choice with the indexing being 0-based: $0 \leq l < N_L$. Figure 1.1 shows an example of the DLDMD network. The layers in the Figure are labeled, sequentially, left to right: Enc in, Enc 0, Enc 1, Enc 2, Enc out, Dec in, Dec 0, Dec 1, Dec 2, Dec out. As will be explained in the next section, passing data through the network involves linear transformations that change the dimension of that data and this is what allows us to “lift” the time-series into the latent space. This realization motivates considering the evolution of each layer individually, which we will revisit in Chapter 3.

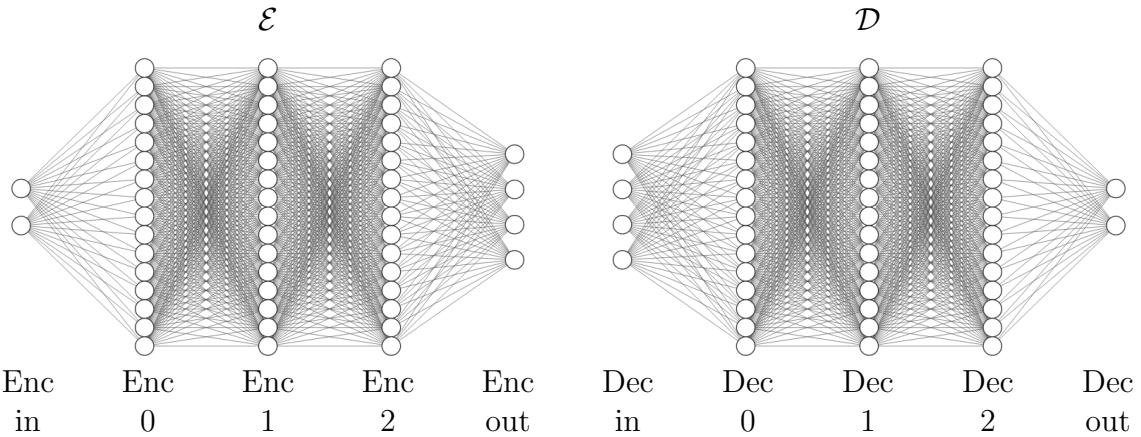


Figure 1.1. Example of DLDMD newtork with $N_S = 2$, $N_O = 4$, and $N_L = 3$ where every hidden layer has 16 neurons. This representation is meaningful in that it gives the name of each layer as they appear from left to right, but it is missing some key network features like the activation functions.

1.5 Statement of Dilemma

Having established DLDMD and having discussed testing it’s limitations via training new models and *visually* qualifying each run as a success or failure, a question is begged: Can we, in an empirical and quantitative fashion, determine whether good training is taking place or otherwise deduce whether our model is well-posed given the hyperparameters? A machine learning model that must learn how to accomplish even a simple task can take quite a long time and plenty of computing resources to complete

it's training and only then can the results be checked to determine whether productive training was taking place as opposed to the optimizer becoming stuck in a local minimum or there being no global minimum at all.

How can this be quantified? A useful starting point would be to examine the weights of the matrices that make up the layers of a model. For dense layers, passing a vector of data $\mathbf{x} \in \mathbb{R}^d$ through a layer L can be written as

$$L(\mathbf{x}; \mathbf{A}, \sigma, \mathbf{b}) = \sigma(\mathbf{A}\mathbf{x} + \mathbf{b}) \quad (1.24)$$

for some matrix $\mathbf{A} \in \mathbb{R}^{N_O \times d}$, vector $\mathbf{b} \in \mathbb{R}^{N_O}$ and (typically nonlinear) activation function $\sigma : \mathbb{R}^{N_O} \rightarrow \mathbb{R}^{N_O}$. For a given layer, σ is a choice of the model, but both \mathbf{A} and \mathbf{b} have entries that are tuned by the optimizer as training takes places. Training takes place on a per epoch basis as the training data is fed into the network and the optimization procedure adjusts the weights relative to the desired metrics, so considering the configuration of these matrices epoch to epoch can help us determine whether they are converging to a set of elements that accomplish the desired task or not. The next state for each layer is dependent on the previous, so a reasonable framing would be as one of a discrete dynamical system of the form

$$Q_{n+1} = P(Q_n) \quad (1.25)$$

where P is the training procedure and Q_n represents the configuration of the network's weights at each epoch n . The initial state Q_0 is random with it's weights being selected from a probability distribution. In the interest of examining convergences, one might consider the following limit

$$\lim_{n \rightarrow \infty} \|Q_{n+1} - Q_n\|_2 \quad (1.26)$$

for some two-norm on the space that Q_n inhabits. While this "one-step" Cauchy convergence will tell us whether the machine is approaching a particular configuration point-wise, this might not actually tell us much of anything else. The changes between epochs is, in some sense, stochastic due to not knowing how the optimizer will update Q between epochs. As such, we propose a more statistical approach wherein we examine how the information content is changing from epoch to epoch. From the stochastic nature of the evolution, we consider the entries of \mathbf{A} and \mathbf{b} for each layer to be drawn from a continuous probability distribution X and examine how it changes epoch to epoch.

CHAPTER 2

KULLBACK-LEIBLER DIVERGENCE

In the field of information theory, we frame information as being represented by a random variable and the values it can take on. For example, consider a normally distributed random variable $X \sim \mathcal{N}(\mu, \sigma^2)$ where μ is the mean or expected value $\mathbb{E}[X]$ of X and σ^2 is the variance $\mathbb{V}[X]$ of X . This is one of the most well understood probability distributions due to countless years of study into its properties, as such we know that if we were to draw a sample for X that we are much more likely to get something close to μ than we are to get something far away from μ . If $\mu = 0$ and $\sigma^2 = 1$, then upon drawing a sample the value 1 is more likely to appear than -37. The level of certainty of the outcome of the draw begs the question of how that certainty can be quantified and conversely, we could instead consider the uncertainty in any given probability distribution. A measure of the uncertainty was proposed in 1948 and this measure is known as *informational entropy* [17] or, simply, entropy. For a continuous distribution with density function $f(x)$, the entropy is given by

$$h[f] = \mathbb{E}[-\log(f(x))] = - \int_X f(x) \log(f(x)) dx \quad (2.1)$$

For the normally distributed X defined above, we have the probability density function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \quad (2.2)$$

and the entropy integral for this evaluates to

$$h[f] = \frac{1}{2} (\log(2\pi\sigma^2) + 1) \quad (2.3)$$

which only depends on the variance. This expression tells us that the entropy of a normally distributed random variable increases as the variance does, graphically this is associated with the flattening of the bell curve which can be interpreted as the values around the mean being less likely to appear relative to values far from the mean. As such, the result of any given sample is less certain; Figure 2.1 demonstrates this.

2.1 Basic Definitions

The Kullback-Leibler Divergence (KLD) is a statistical distance between a pair of probability distributions which measures how different a distribution P is from a

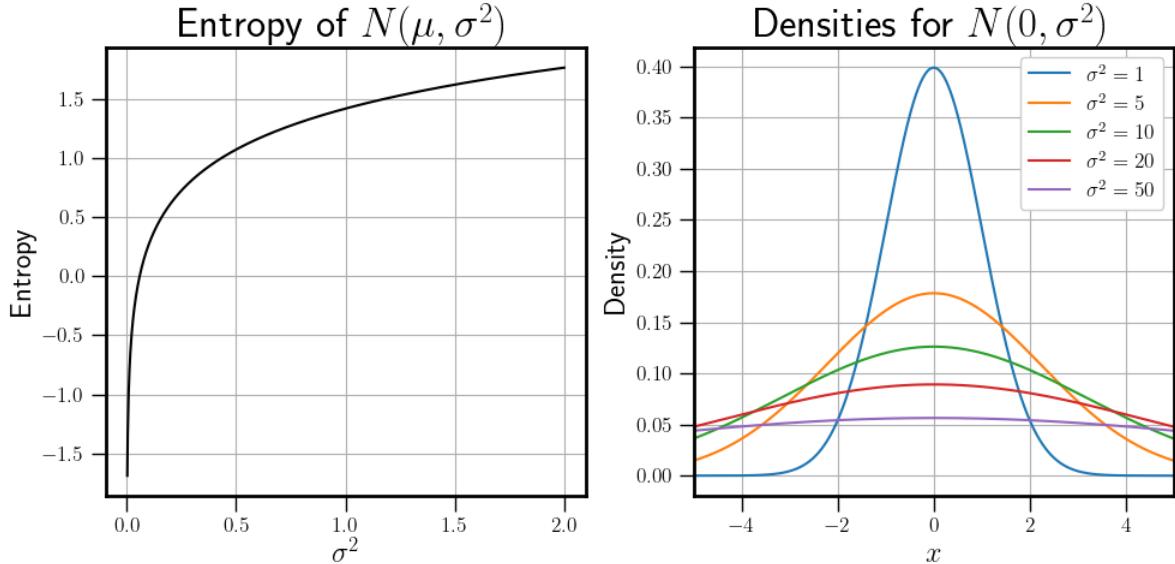


Figure 2.1. Entropy and density function change with respect to σ^2 . The base of the logarithm shown here is e , but the base itself is irrelevant due to the shared properties of all logarithms: that they increase monotonically and unbounded. As the variance increases we see higher entropy in the left plot and a flattening of the probability density in the right plot.

reference distribution Q . A simple interpretation of the divergence of P from Q is the expected excess surprise from using Q as a model when the actual distribution is P [10]. If P and Q are discrete distributions defined on the probability space X , then the KLD of P with respect to Q is given by

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log(P(x)/Q(x)) \quad (2.4)$$

In other words, it is the expectation of the logarithmic difference between the probabilities P and Q , where the expectation is taken using the probabilities P . As a “distance”, the KLD is non-negative and 0 when $P = Q$ almost everywhere. Unlike metrics, it is not symmetric and does not satisfy the triangle inequality. In order to deal with something that is, at least, symmetric we are using the symmetric KLD (SKLD) which is defined as

$$D_{SKL}(P, Q) = \frac{1}{2} (D_{KL}(P \parallel Q) + D_{KL}(Q \parallel P)), \quad (2.5)$$

which is the average of the KLD for P with respect to Q and for Q with respect to P . In our cases, we are dealing with a continuous random variable with probability density functions p and q , for which the corresponding KLD formula is

$$D_{KL}(P \parallel Q) = \int_X p(x) \log(p(x)/q(x)) dx \quad (2.6)$$

As an analogue of the discrete case, the same simple interpretations about the formula apply. The reason we can consider this a “distance” is that the KLD is non-negative, $D_{KL}(P \parallel Q) \geq 0$, which is known as Gibbs inequality and only equals 0 when $P = Q$ almost everywhere. As such, the distance between P and Q is 0 when they are the “same” distribution and some positive number if they differ. For example, consider the random variables $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ with corresponding density functions given by equation 2.2; the divergence between the distributions is

$$D_{KL}(p \parallel q) = \frac{1}{2} \left(\ln \left(\frac{\sigma_2^2}{\sigma_1^2} \right) + \frac{\sigma_1^2}{\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{\sigma_2^2} - 1 \right), \quad (2.7)$$

Letting $\mu_1 = \mu_2$, $\sigma_1^2 = 1$, and allowing σ_2^2 to vary, we can write

$$D_{KL}(p \parallel q) = \frac{1}{2} \left(\ln \sigma_2^2 + \frac{1}{\sigma_2^2} - 1 \right), \quad (2.8)$$

Figure 2.2 illustrates the KLD between the example normal distributions.

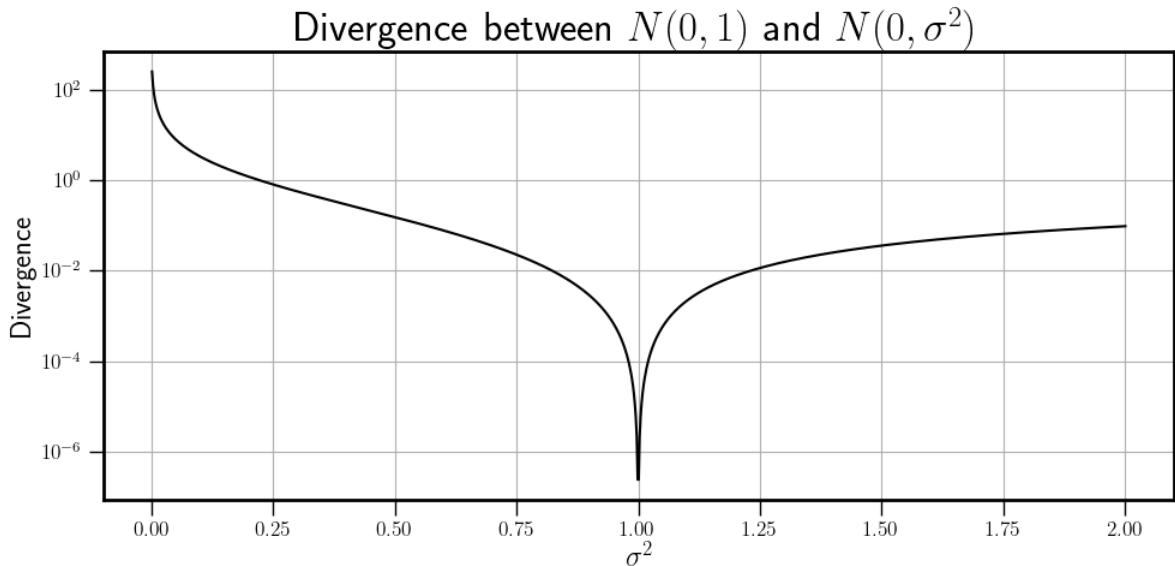


Figure 2.2. Kullback-Leibler Divergence example with two normal distributions. As expected, the divergence decreases to 0 as $\sigma^2 \rightarrow 1$ from either direction.

2.2 How to build distributions (Non-parametric statistics)

The question remains about how we obtain the probability distribution of the random variable that represents the evolution of each layer of an ML model from

measured data. Many techniques exist, but we are using Kernel Density Estimation (KDE) [13, 12]. The most basic idea behind KDE is using a bin centering method and a smoothing factor, called a kernel, to approximate a probability density f from measured data in the form of a histogram. This is accomplished with the *kernel density estimator* for f given by

$$\hat{f}(x; K, h) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right), \quad (2.9)$$

where K is the chosen kernel, h is what's called the bandwidth parameter, and $\{x_i\}_{i=1}^n$ is the data that generates our histogram. Per Epanechnikov, the choice of kernel does not provide much statistical significance [4]; but the choice of the bandwidth parameter h is very crucial for finding a density estimate that approximates the underlying density function appropriately and can be thought of as an analogue of the bin width for the affiliated histogram. Choosing an optimal h depends on the given data and the common method, Silverman's rule of thumb, works on the assumption that the density function of the data is unimodal and close to normal. For general data, on which no assumptions of normality are made, a more general method is found in the Improved Sheather-Jones algorithm (ISJ) [2]. The ISJ seeks to minimize the asymptotic-mean-integrated-squared error (AMISE) of the estimator \hat{f} with respect to h , which is given by

$$\text{AMISE}(\hat{f}) = \frac{h^2}{4} \|f''\|^2 + \frac{1}{2n\sqrt{h\pi}} \quad (2.10)$$

and depends on the unknown quantity $\|f''\|$. The analysis that [16] uses to derive equation 2.10 relies on a few assumptions, such as the existence of f'' , which may not always apply to an arbitrary density. The ISJ uses an iterative method based on the data and kernel function to approximate $\|f''\|$ numerically to assuage that issue and arrive at the best bandwidth for the given data. See Figure 2.3 for an example density found using KDE and the ISJ algorithm.

2.3 Measuring entropy flow

At each epoch n of training for each layer l in the encoder \mathcal{E} and decoder \mathcal{D} networks of the DLDMD algorithm, we generate an affiliated matrix of weights $\mathbf{W}_{l,\mathcal{E}}^{(n)}$ and $\mathbf{W}_{l,\mathcal{D}}^{(n)}$, respectively. Should any given training cause the machine to converge to a fixed state, then for each layer we should have that

$$\mathbf{W}_{l,\mathcal{E}}^{(n)}, \mathbf{W}_{l,\mathcal{D}}^{(n)} \rightarrow \mathbf{W}_{l,\mathcal{E}}^*, \mathbf{W}_{l,\mathcal{D}}^* \text{ as } n \rightarrow \infty \quad (2.11)$$

Consequently, we can characterize any set of weights via the formulas

$$\mathbf{W}_{l,\mathcal{E}}^{(n)} = \mathbf{W}_{l,\mathcal{E}}^* + \mathbf{W}_{l,\mathcal{E}}^{(n),f}, \quad \mathbf{W}_{l,\mathcal{D}}^{(n)} = \mathbf{W}_{l,\mathcal{D}}^* + \mathbf{W}_{l,\mathcal{D}}^{(n),f}, \quad (2.12)$$

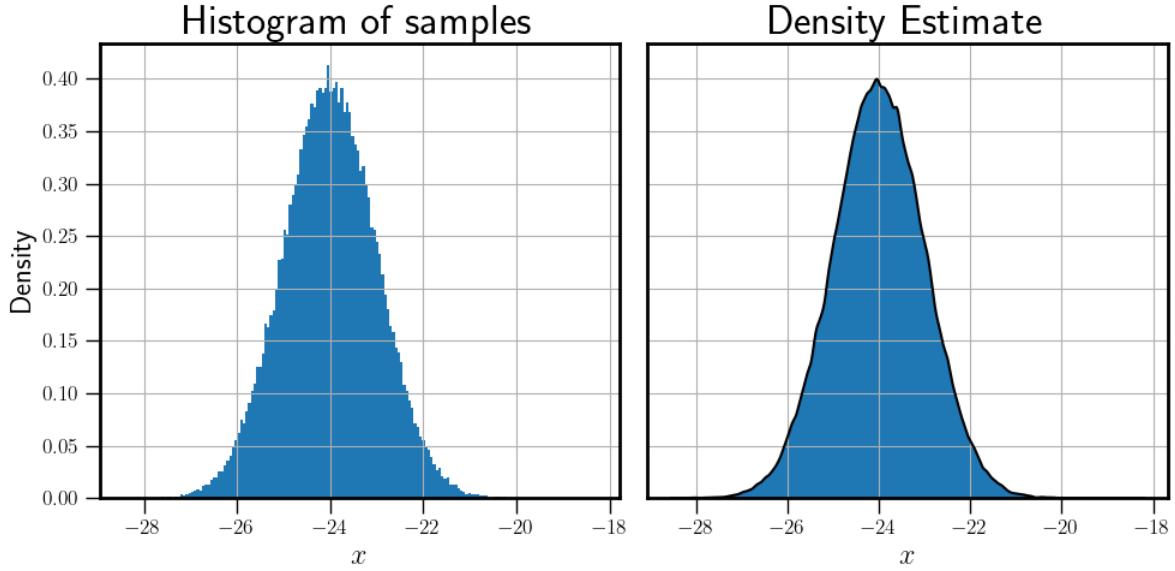


Figure 2.3. Example of a density function found using KDE, its shape coincides with that of the histogram generated from the same samples.

where $\mathbf{W}_{l,\mathcal{E}}^{(n),f}$ and $\mathbf{W}_{l,\mathcal{D}}^{(n),f}$ can be thought of as fluctuations from the steady state. These fluctuations are what we're most interested in studying. Using first-order differencing, we can study the affiliated detrended matrices $\delta\mathbf{W}_{l,\mathcal{E}}^{(n)}$ where

$$\delta\mathbf{W}_{l,\mathcal{E}}^{(n)} = \mathbf{W}_{l,\mathcal{E}}^{(n+1)} - \mathbf{W}_{l,\mathcal{E}}^{(n)} \quad (2.13)$$

$$= \mathbf{W}_{l,\mathcal{E}}^{(n+1),f} - \mathbf{W}_{l,\mathcal{E}}^{(n),f} \quad (2.14)$$

The definitions for the decoder weights are identical, and thus omitted. We can consider $\delta\mathbf{W}_{l,\mathcal{E}}^{(n)}$ to be the deviation from the steady state and with it in hand we flatten it's data and use KDE to generate an affiliated empirical probability distribution $p_{l,\mathcal{E}}^{(n)}(w)$. With these distributions we now find the KLD between consecutive distributions to generate the following sequence:

$$D = \left\{ D_{KL} \left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E-2} \quad (2.15)$$

which we can interpret to be the change in information between consecutive fluctuations from the steady state.

2.4 Implementation Details

The devil is in the details with respect to how this analysis is carried out, and we go over the important ones here. After the training of each model for N_E epochs, the data that we have to play with is

$$\mathbf{W}_{\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_L}, \mathbf{W}_{\mathcal{D}} = \left\{ \mathbf{W}_{l,\mathcal{D}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_L}, \quad (2.16)$$

which are the sets of weights of the layers of the encoder and decoder. Note that both sub-networks have the same number of layers N_L and only differ by the dimension of their inputs and outputs. The following procedures are identical for both sets, so we only reference the encoder set from here on out. For each layer l , the set of detrended matrices are computed

$$\delta\mathbf{W}_{l,\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n+1),f} - \mathbf{W}_{l,\mathcal{E}}^{(n),f} \right\}_{n=1}^{N_E-1}, \quad (2.17)$$

these matrices are flattened and their entries are used as the data for KDE. The KDE implementation used is that of KDEPy (<https://pypi.org/project/KDEPy/>), which implements several algorithms for KDE in Python including the bandwidth computation of ISJ and a very fast implementation of the KDE that uses the Fast Fourier Transform. The kernel used is the Epanechnikov kernel, defined as

$$K(x) = \begin{cases} \frac{3(1-x^2)}{4}, & |x| < 1 \\ 0, & |x| \geq 1 \end{cases}, \quad (2.18)$$

which is optimal in the mean-square error sense. However, the choice of kernel is not a statistically significant one [4] and we choose this kernel for the convenience of it having finite support and non-vanishing first and second derivative on that support. The ISJ algorithm is used to determine the appropriate bandwidth for each set of data. These bandwidths are also being saved to see if the optimal bandwidth with respect to each data set might also contain some instructive information for quantification. From this, we obtain the set of density estimates for each detrended matrix

$$p_{l,\mathcal{E}} = \text{KDE}(\delta\mathbf{W}_{l,\mathcal{E}}) = \left\{ p_{l,\mathcal{E}}^{(n)}(w) \right\}_{n=1}^{N_E-1} \quad (2.19)$$

where w is the weight value itself and $p_{l,\mathcal{E}}(w)$ is the approximate density associated with that weight value. For the sake of our discretization of this data, we use 5001 linearly spaced points to represent the weights of each distribution from the smallest weight to the largest. The number of discrete points is also of some importance for the next step; given these density approximations, we can now compute what we are truly interested in, the relative entropy of consecutive densities and the associated change in information:

$$D_l = \left\{ D_{SKL} \left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E-2} \quad (2.20)$$

Unlike in the first mention of entropy flow, we instead use the SKLD instead of the KLD. This consideration simplifies our analysis by disregarding the asymmetric nature of KLD and consequences of this choice will be discussed in Chapter 4. Given the

discretized nature of the densities, we turn to numerical integration to compute the integral in Equation 2.6. For the sake of an apples to apples comparison, we make the assumption that all densities associated with a given layer have the same support and use the 5001 discretized points with a sum scaled by the spacing between those points for the integral approximation. Each set of these divergences corresponds to a layer in the full network, and we can now use the divergences to characterize how the information is changing with respect to the steady state. Figure 2.4 shows an example of the divergence of the detrended data for a single layer over the whole training.

We are interested in what these divergences tell us about information flow in the model's weights and seek to identify specific classifiers of good training. That is the ultimate goal of this thesis: Can we find quantifiable attributes of the network that behave one way for good training and in a distinguishably different way for bad training? Finding fittings of the data of the divergence data is the most reasonable place to start and trends will provide for easier classification. Any overall trend is more important to our analysis than lone data points, so to avoid transient behavior skewing any fittings we ignore the first 20% of the divergences computed. The values span several orders of magnitude, so a simple fitting would certainly fail and we instead attempt a linear fit on the base 10 logarithm of the data. This corresponds to an exponential fit in the original domain of the divergences with slope of the linear trend being the approximate growth/decay rate of the information flow from the machines steady state. The slope intercept is not so important for this reason, being only a constant that modifies the order of magnitude without changing the trend, so for the time being we ignore it. In the interest of trying to find a classifier of good training, we compute the average and variance of the slopes of the linear fits of the layer's divergence data. The backbone of our results is to compare these averages for each latent dimension to the loss curves and phase portraits from the training for each dimension. We have written code that implements all of these, and they can be found on GitHub at https://github.com/JAGDiaz/Thesis_Analysis_Scripts.

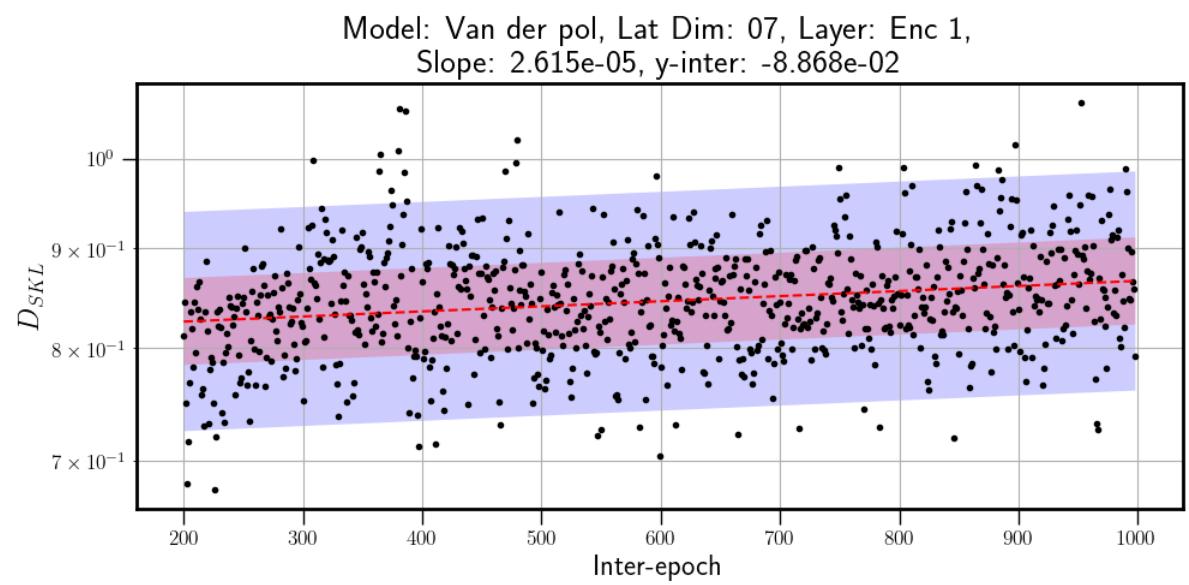


Figure 2.4. Example of linear fitting of divergence data for one of the models we examine in Chapter 3.

CHAPTER 3

RESULTS

From experimentation by [1], we know that some sets of hyperparameters yield good reconstruction of the structure of the phase space of each of the dynamical systems in question and that some do not. For the sake of testing our method of measuring information change, we elect to focus on changing only the lifting or latent dimension of the observables we wish to use for the DMD algorithm. After conducting multiple trainings with latent dimensions ranging from $N_O = 2$ to $N_O = 10$ for the Duffing and Van der Pol oscillators, the algorithm detailed in Chapter 2 was implemented. The initial hope of our analysis is that data points that our algorithm generates will show clear trends that allow us to easily differentiate the optimal parameter choices for the model from suboptimal or even bad parameter choices. Given the model's task of learning an embedding and find optimal observables in the latent space, we additionally surmised that we would see the most variability in the model in what we call the edge layers: Enc in, Enc out, Dec in and Dec out. The other layers in the model certainly accomplish something as the model trains, but the dimension changes that help EDMD along are accomplished at the edge layers, so we expect them to evolve differently than the hidden layers.

3.1 The Duffing Oscillator

The Duffing oscillator is a non-linear second-order differential equation used to model certain damped and driven oscillators, whose general form is given by

$$\ddot{x} + \delta\dot{x} + \alpha x + \beta x^3 = \gamma \cos(\omega t) \quad (3.1)$$

There are many sets of coefficients for which many types of behavior are exhibited by the system, but for our purposes we are only considering the following configuration:

$$\ddot{x} - x + x^3 = 0 \quad (3.2)$$

which when turned into a system of first-order equations by introducing $y = \dot{x}$ is

$$\begin{cases} \dot{x} = y \\ \dot{y} = x - x^3 \end{cases} \quad (3.3)$$

The interesting attributes of this system are that it has three fixed points, two of which are non-linear centers with the third being a saddle point with homoclinic

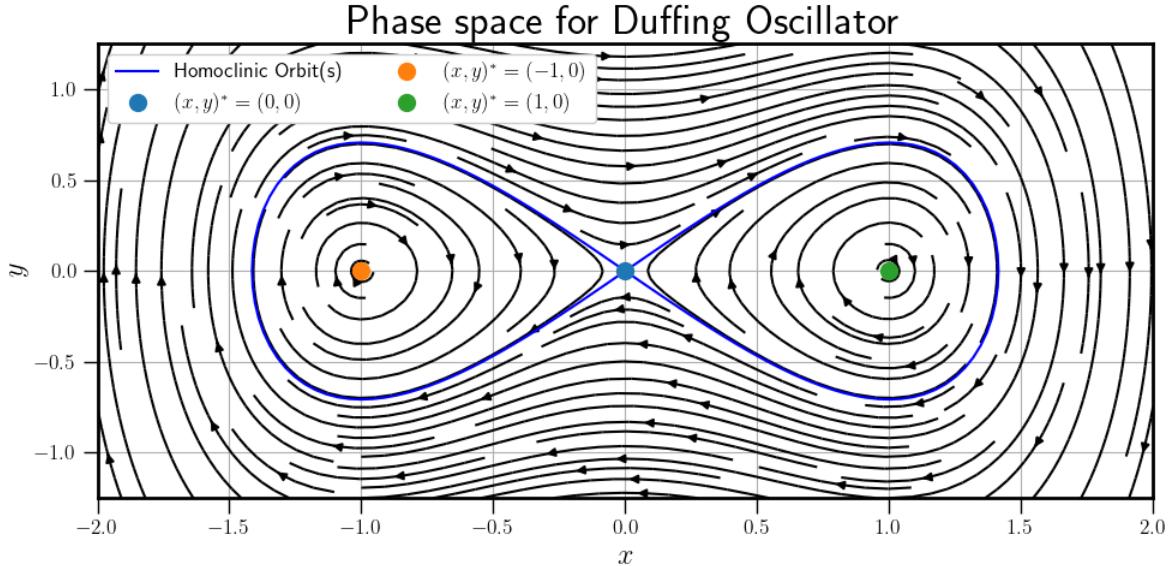


Figure 3.1. Duffing phase space.

Table 3.1. Hyperparameters of the DLDMD algorithm held constant for each training run on Duffing.

Hyperparameter	Value
Initial Conditions	15000
Training proportion	66.66%
Validation proportion	19.98%
Testing proportion	13.36%
Optimizer	Adam
t_0	0
Δt	0.05
t_f	20
N_E	1000
Enc/Dec layers	3
Enc/Dec hidden activation	ELU
Enc/Dec output activation	identity
Neurons/layer	128
Weight initializer	$\mathcal{N}(0, .1, 2)$
Bias Initializer	0
Learning Rate	10^{-3}
a_1, a_2, a_3	1
a_4	10^{-14}

connections that encompass the other two fixed points. Figure 3.1 shows a graphical representation of the phase space of our Duffing oscillator. The homoclinic connection

itself provides an interesting challenge topologically in that it divides the phase space into distinct regions that affect the end behavior of the system depending on the initial condition [18]. Table 3.1 describes the hyperparameters used for the Duffing Oscillator that were left unchanged for each run; note that the distribution given as $\mathcal{N}(\mu, \sigma^2, n)$ is the truncated normal distribution, which is a normal distribution in which all values drawn that are more than n standard deviations from the mean are discarded.

A few notes on the trainings themselves are worth mentioning here. No learning rate scheduling is used, so the learning rate with the Adam optimizer is held constant for all epochs. The activation function Exponential Linear Unit (ELU) is used instead of the more standard Rectified Linear Unit (ReLU) due an error that occurs in the numerics of DLDMD for latent dimension choices greater than three. ELU is a parameterized activation function whose definition is given in Table 3.2 and is closely related to ReLU; in fact, if the parameter α is set to 0 then ELU and ReLU are identical. For the sake of having similar performance for ELU with respect to ReLU, we choose $\alpha = .01$. As stated in the caption for Table 3.2, ReLU can lead some models to “kill” the weights associated with a given neuron by causing them to go to 0. Consequently, this can cause the data being fed into the model to have more zeros than is useful for EDMD and the matrices in that step become ill-conditioned. This happens to Duffing using ReLU if the latent dimension is three or greater and the algorithm subsequently ends with an error. After experimentation, it was found that using ELU instead circumvents this issue that ReLU raises without significantly affecting model training. As a final note, the training for this model (and Van der Pol) across all latent dimensions was done on the same data set with the same subsets chosen to be the training, test, and validation sets. We do this to reduce the number of variables down to the latent dimension N_O alone.

ReLU	ELU
$\sigma(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$	$\sigma(x; \alpha) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x \leq 0 \end{cases}$

Table 3.2. ReLU activation function vs ELU activation function. Unlike ReLU, ELU is negative for $x < 0$. For some models, ELU can be advantageous if ReLU causes many of the neuron to “die” during training which means the weights associated with that neuron become 0, and those neurons do not contribute anything to output of the machine. Other activation functions can assuage this problem, but ELU can be made arbitrarily close to ReLU and so is quite popular.

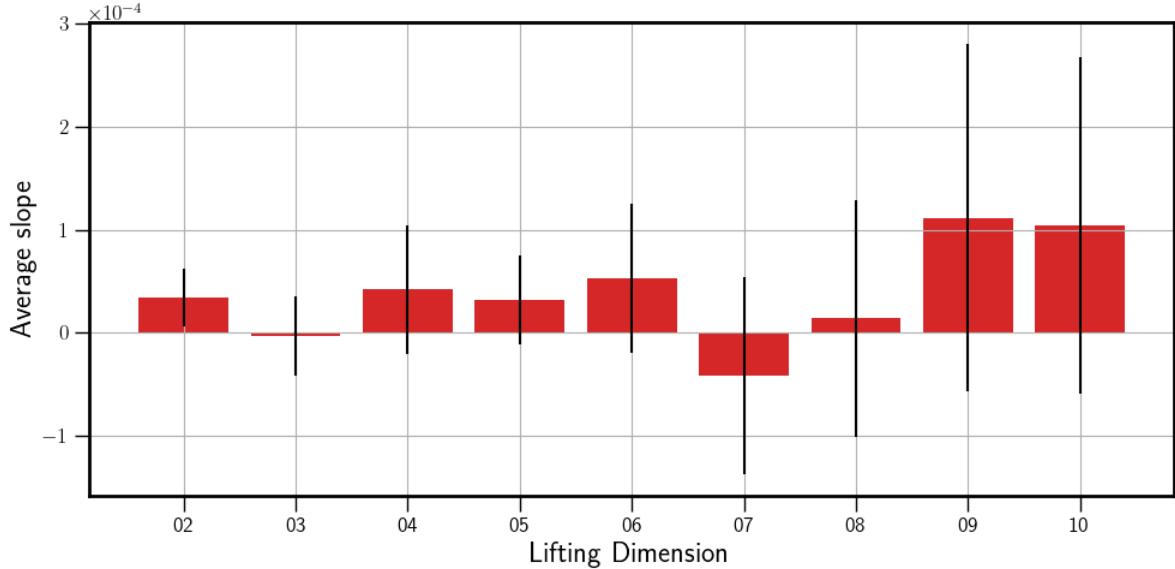


Figure 3.2. Duffing average slopes of linear fits with standard deviations. This plot illustrates what we want to find. Each latent dimension gives us a different behavior and we can readily see how the information must be changing during training.

Having established this, we now consider the results of our analysis. Figure 3.2 shows the average of the slopes of the linear fits across all layers for each epoch. What we can see plainly is that for latent dimensions 3 and 8, the average slope is small in magnitude with a small to medium standard deviation relative to the scale with the other latent dimensions showing much larger average slopes and higher variance. We may contrast this with the losses in Figure 3.7 for each latent dimension, which seems to show that those latent dimensions should be the optimal choices with all other choices being more or less equivalent. These phase space representations are very close indeed, however, which motivates examining other metrics. If we examine Figure 3.4, which shows the average bandwidth from the ISJ algorithm across all layers for a given latent dimension per epoch, we can see that each latent dimension choice seems to fall into the same band between $4 \cdot 10^{-6}$ and $6 \cdot 10^{-6}$. From this we can deduce that the overall trend in computed bandwidths from the ISJ algorithm is not significantly different for any latent dimension; so it seems that this may not be a useful value for differentiating between good parameter choices. If we examine Figure 3.3, we see that considering loss curves alone can be unreliable, and a contrast with Figure 3.7 is quite instructive when trying to recognize why you may want to consider more than just loss curves when evaluating the utility of a ML model. Even so, we see that latent dimension 3 has a better loss curve, a faithful phase space forecasting and a small

average slope with respect to Figure 3.2 and we conclude that this is a good indication of successful training.

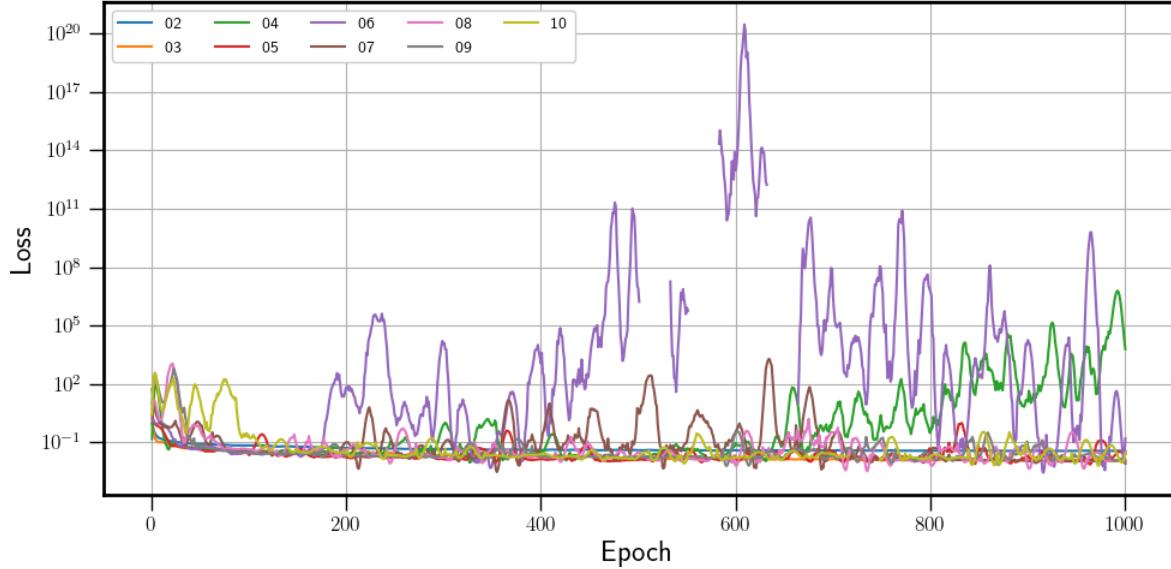


Figure 3.3. Duffing loss plots, smoothed to accentuate trends.

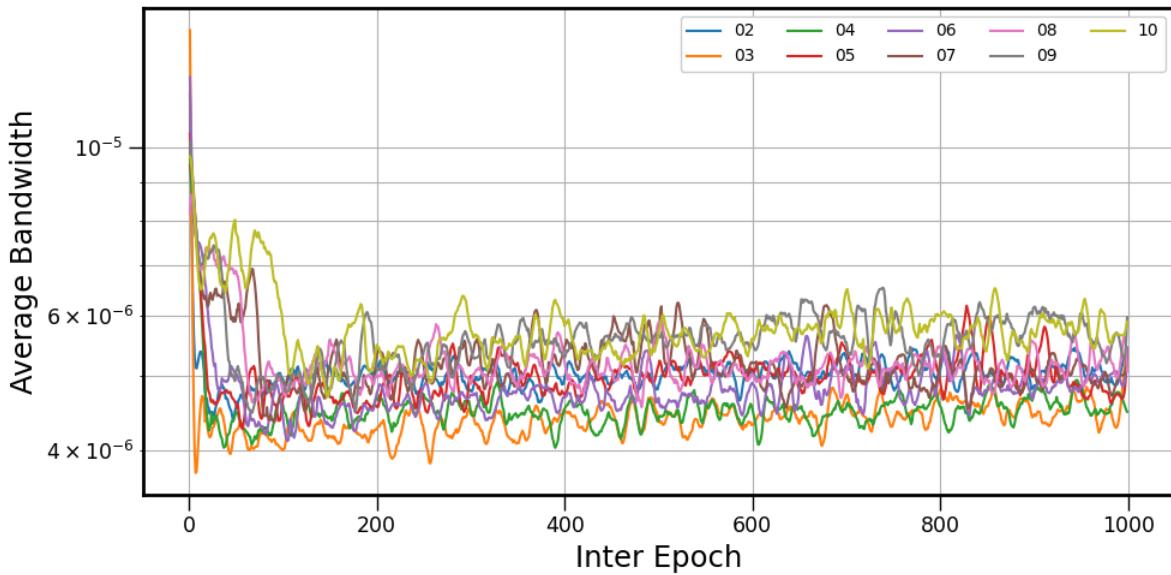


Figure 3.4. Duffing Average bandwidth.

For a more granular view of the individual fittings for each layer of the model, we have Figure 3.5. Much to our expectation, the plots for the edge layers have slightly

different behavior than for the hidden layers; generally, the linear trends have a smaller slope and y -intercept further from 0. We can interpret this as being indicative of having an information change trend that is further from the theoretical steady-state than for the other layers of the model. We have come to associate this with the kind of expressivity that is desireable for ML models to have in that they are able to cope with the kind of variability that real world data could exhibit. The ultimate conclusion we are able to draw is that DLDMD seems to adequately train and predict future states of Duffing for latent dimensions 3, 4, 5, and 6.

3.2 The Van der Pol Oscillator

The Van der Pol oscillator is a non-conservative oscillator with non-linear damping, and models a more specific kind of circuit than Duffing does. It's general form is given by

$$\ddot{x} - \mu(1 - x^2)\dot{x} + x = 0 \quad (3.4)$$

Much like Duffing, varying μ can give different and interesting phase space dynamics. For our purposes we fix it to $\mu = 1.5$. With this, the equivalent system of first order equations is

$$\begin{cases} \dot{x} = y \\ \dot{y} = 1.5(1 - x^2)y - x \end{cases} \quad (3.5)$$

Unlike Duffing, this system has a single unstable fixed point at the origin and all nontrivial orbits are attracted to a limit cycle that surrounds the origin. Figure 3.8 shows a graphical representation of the phase space of our Van der Pol oscillator. Much like Duffing, we do have a curve that separates the phase space into regions of different behavior (the limit cycle), but since all orbits approach the limit cycle there shouldn't be much variability in end behavior with respect to initial conditions [18]. We would expect training on this system to be better overall due to the fact that all trajectories are essentially going to same place unlike the more complex behavior of Duffing. The hyperparameters for the Van der Pol training are identical to those of Duffing with the exception of the activation function; which is, as mentioned in Section 3.1, changed to ReLU. This difference is minor enough that a Van der Pol hyperparameters table is omitted.

Figure 3.9, shows the average and standard deviation of the slopes for the Van der Pol training per latent dimension choice. Relative to Figure 3.2 for Duffing, we have a more uniform decrease in the magnitude of the average slope as the latent dimension increases and higher variances in general. Based on the phase plane results in Figure 3.14, we should expect 6 and 9 to correspond to the best training so it seems that a

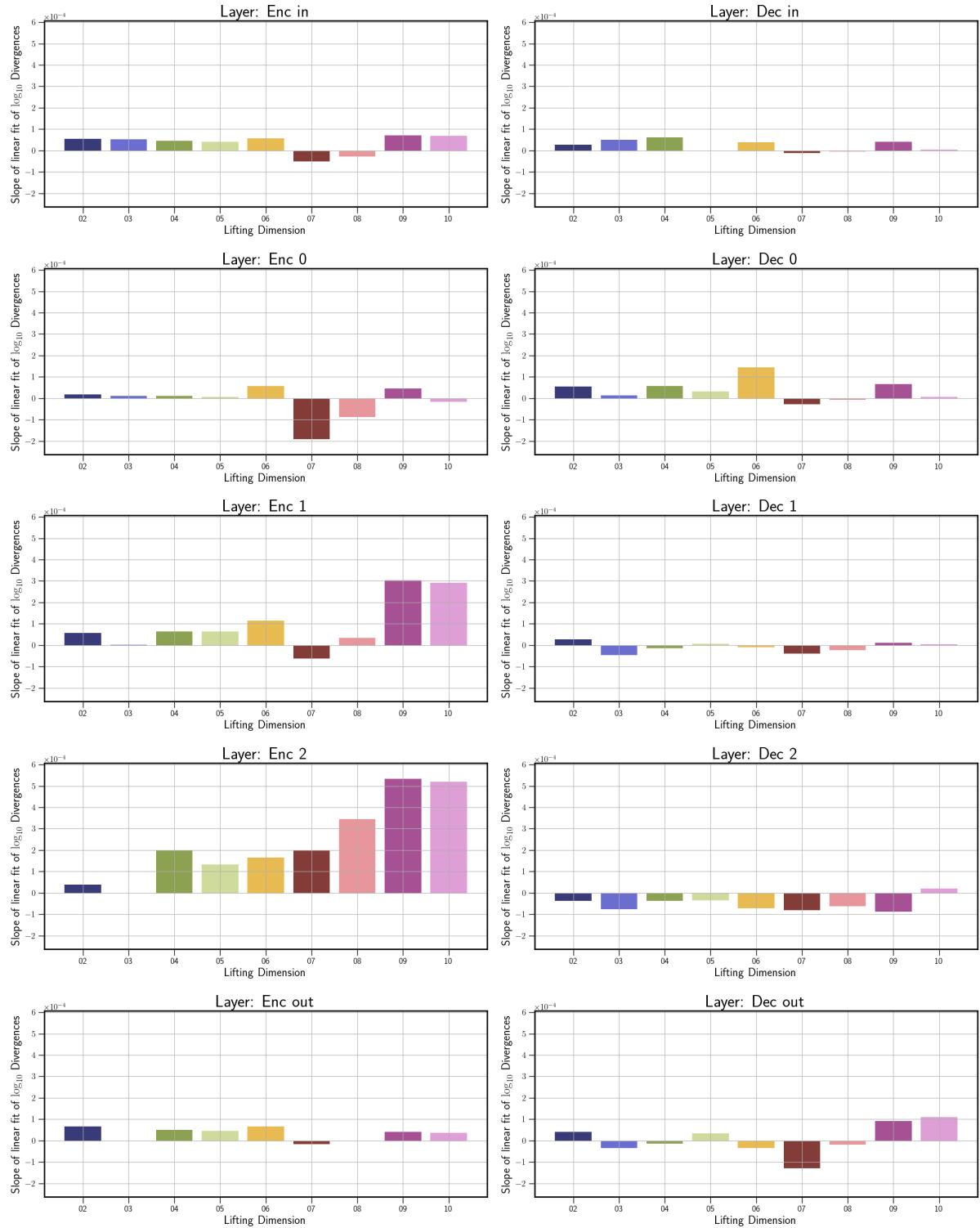


Figure 3.5. Duffing slope of linear fits by layer.

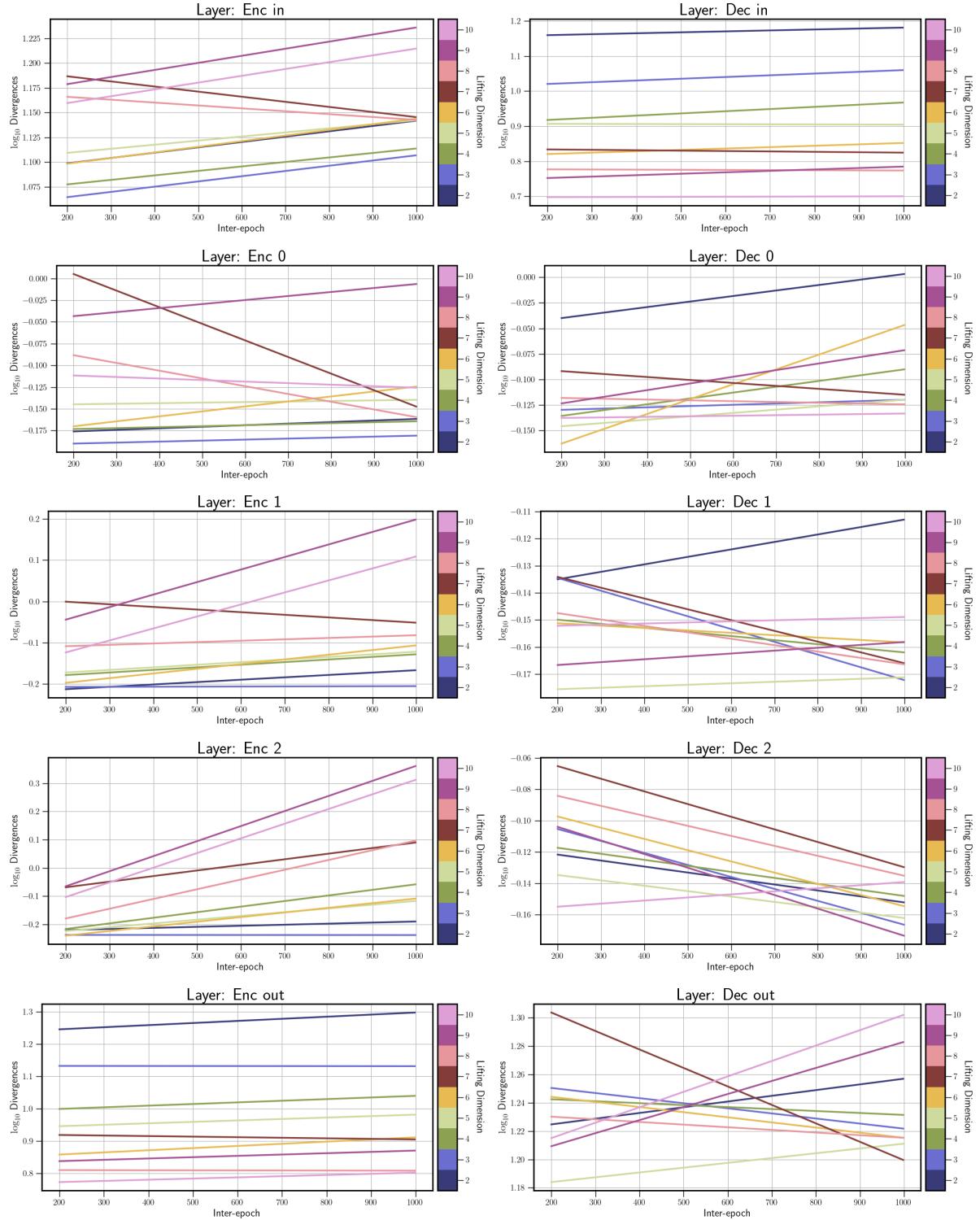


Figure 3.6. Duffing linear fits by layer.

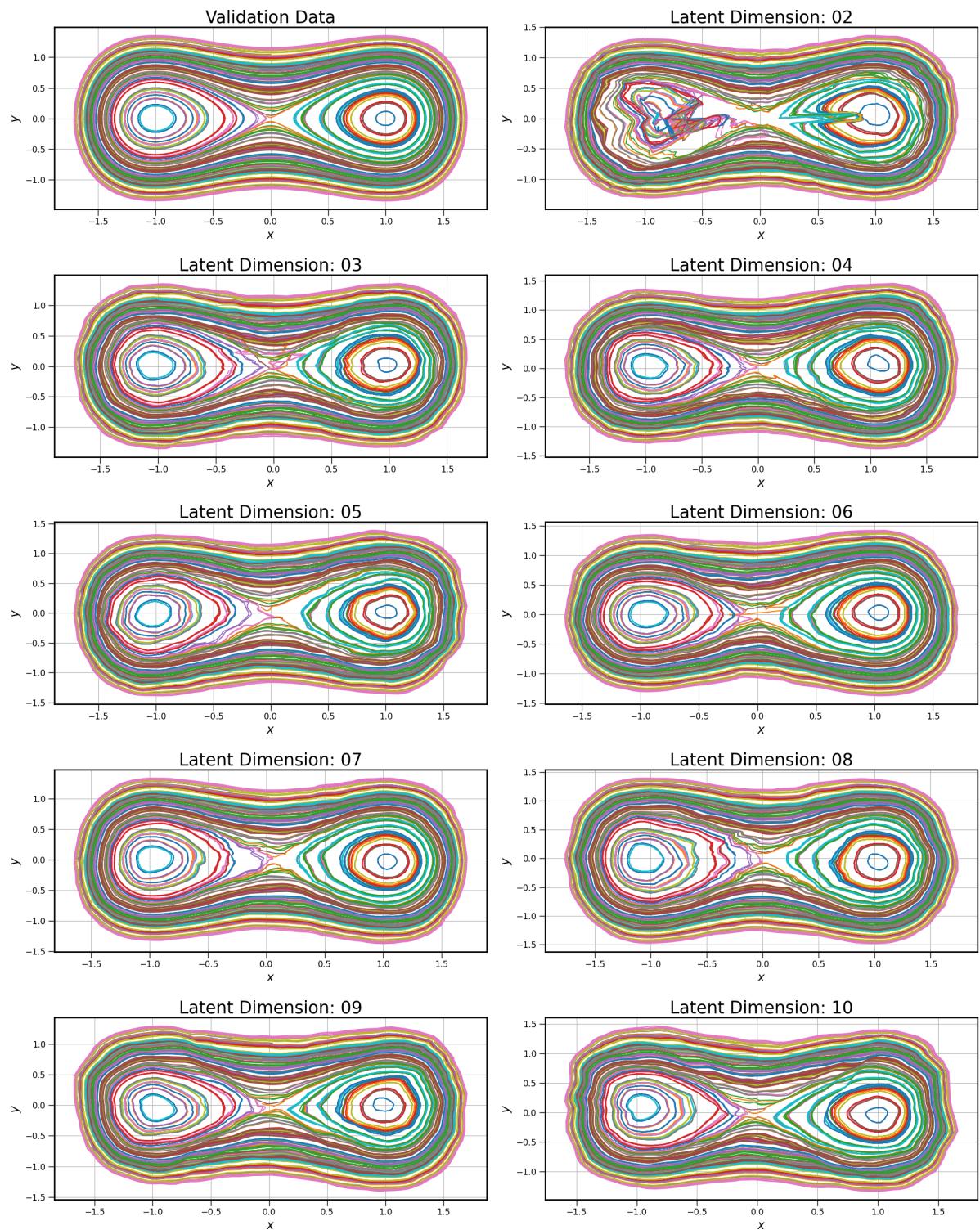


Figure 3.7. Duffing DLDMD results by latent dimension.

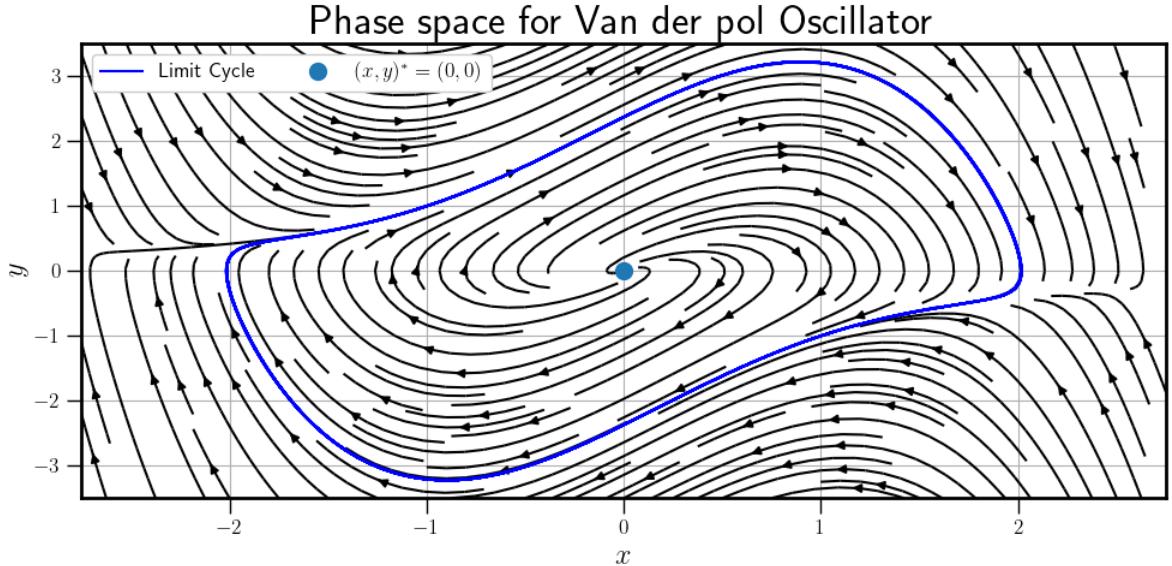


Figure 3.8. Van der Pol phase space.

small in magnitude slope is indicative of good training and this is consistent for some of the loss curves in Figure 3.10. However, the loss curves of each latent dimension choice aside from 2 fall on one another and the average bandwidths in 3.11 do as well. We can see in Figures 3.12 and 3.13 that like in Duffing, the edge layers evolve differently than the hidden layers and we see that in their having larger y -intercepts and more variation in their slopes as well. With this more granular view, we can also see why all the average slopes are negative, which is not the case for Duffing. The exact interpretation of this will be discussed later, but we have come to see this is as the trend of deviation from the steady growing smaller as the model trains and a consequence of the simpler phase space of Van der Pol.

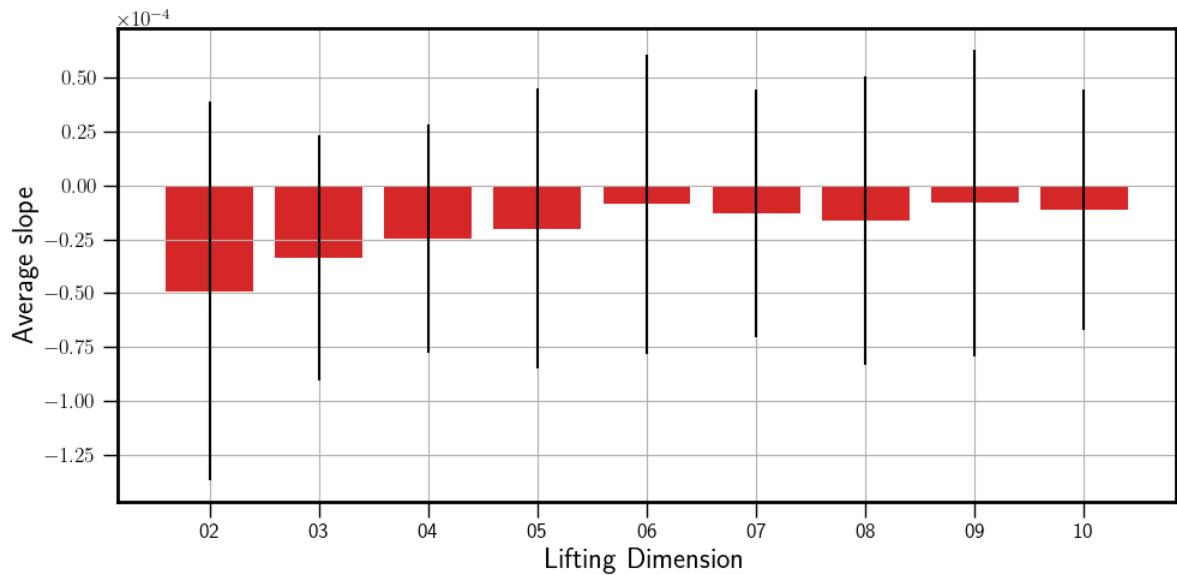


Figure 3.9. Van der Pol average slopes with standard deviations.

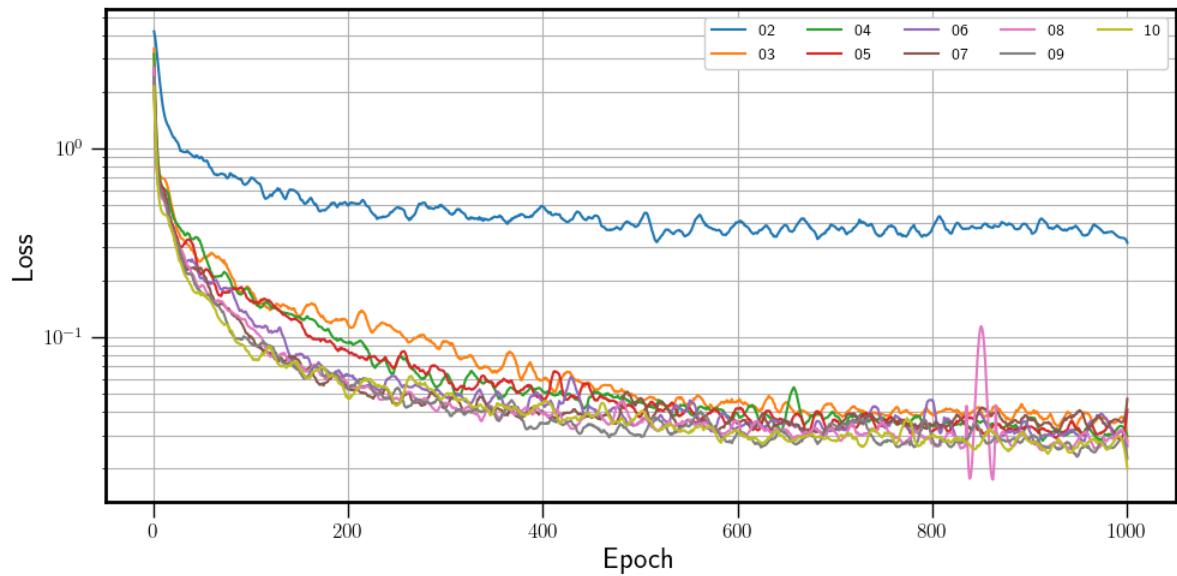


Figure 3.10. Van der Pol loss plots, smoothed to accentuate trend.

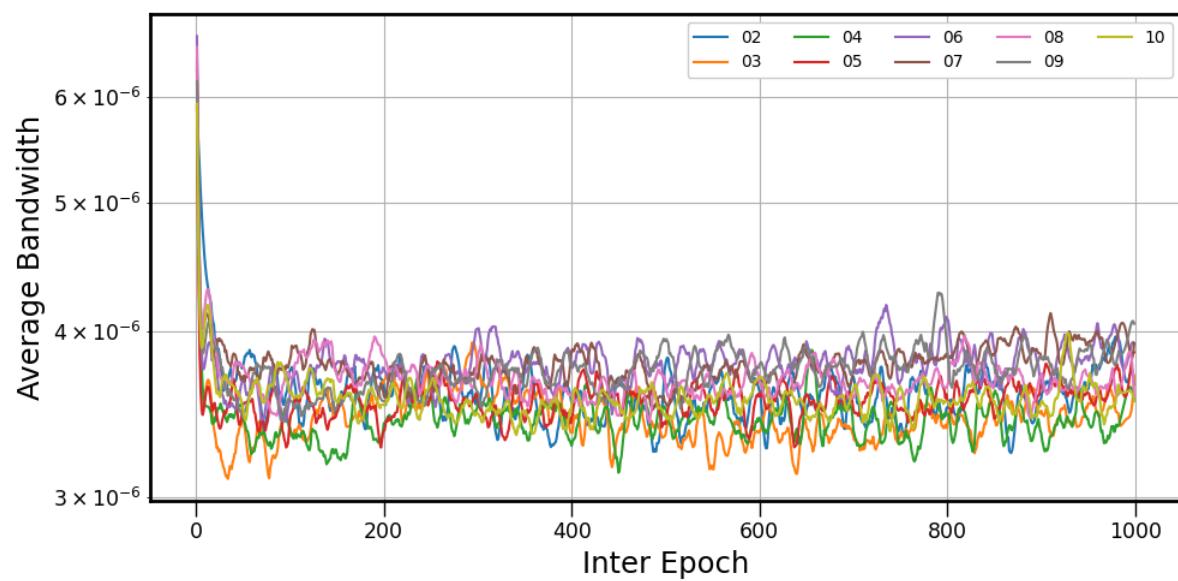


Figure 3.11. Van der Pol Average bandwidth.

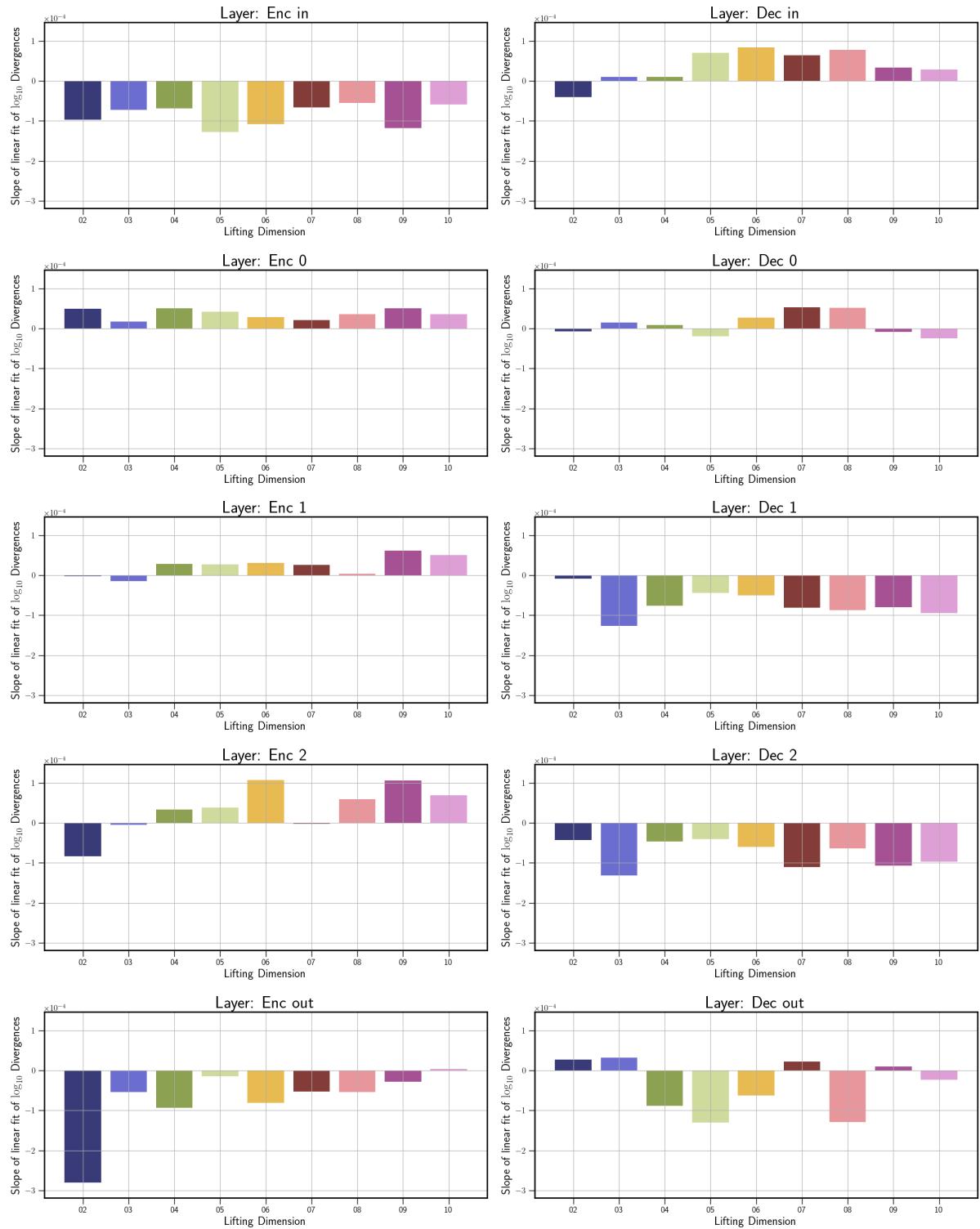


Figure 3.12. Van der Pol slope of linear fits by layer.

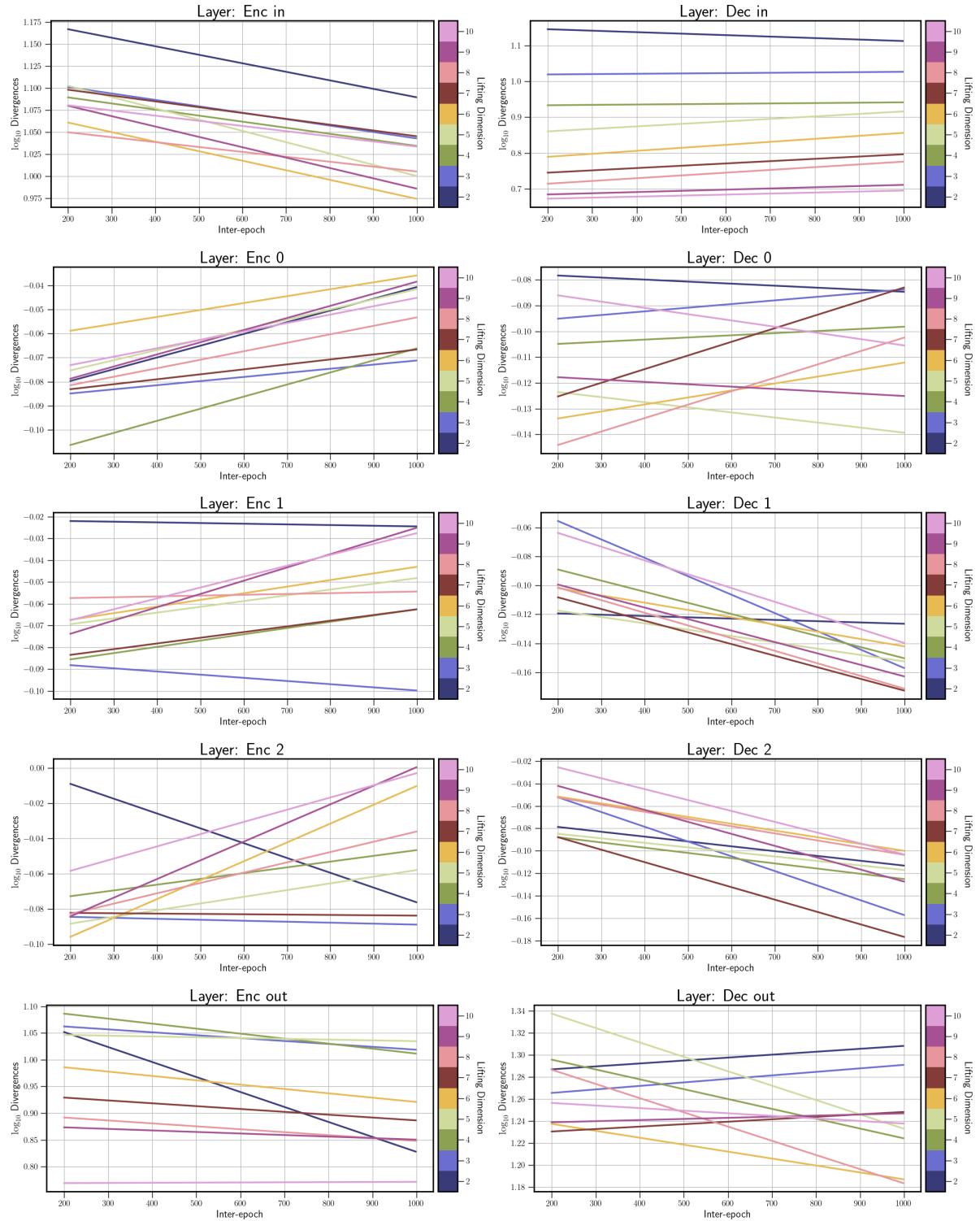


Figure 3.13. Van der Pol linear fits by layer.

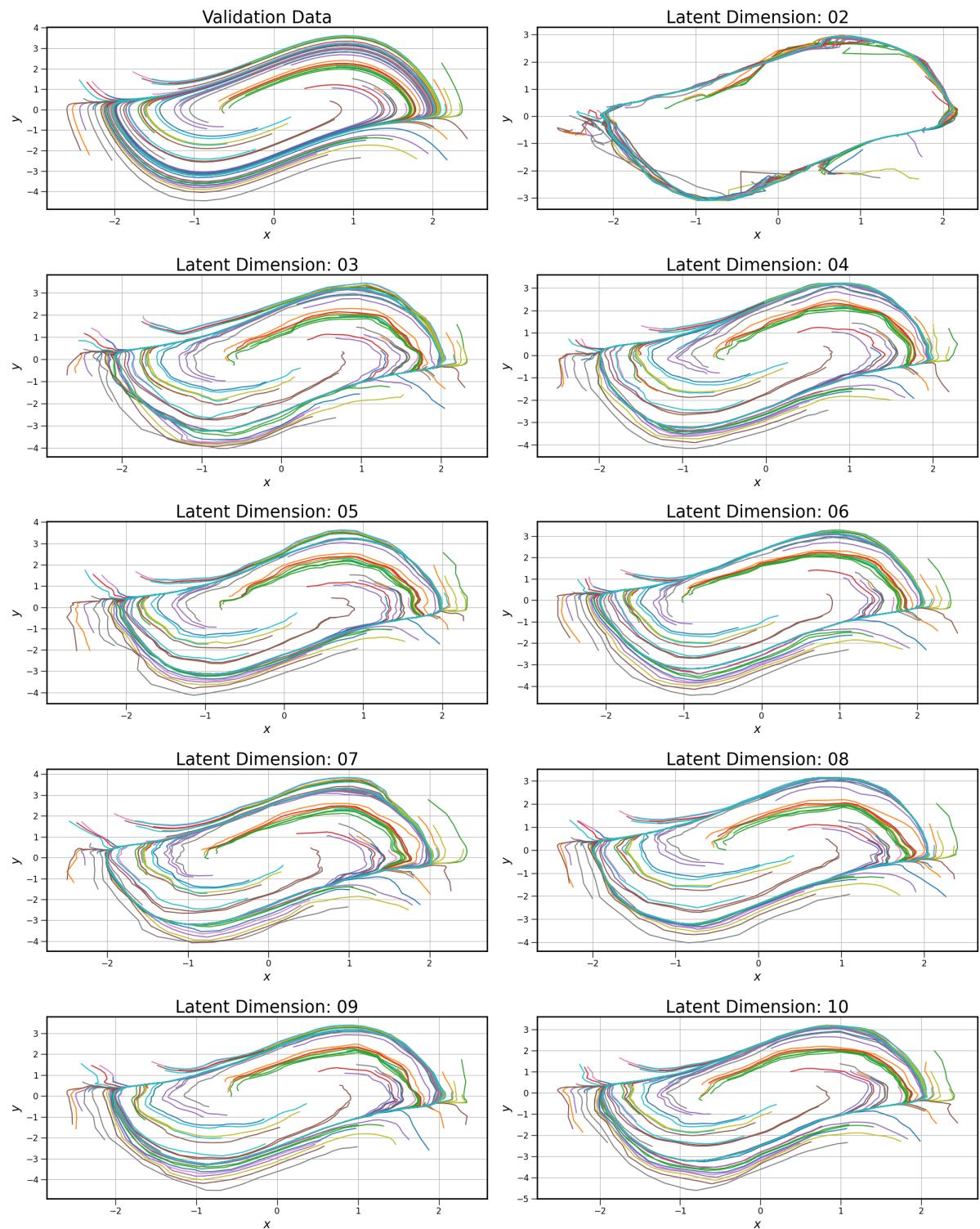


Figure 3.14. Van der Pol DLDMD results by latent dimension.

CHAPTER 4

DISCUSSION

Having presented these results, we briefly discuss some of the consequences we can deduce from them and consider some interpretations of the fittings. The divergences represent the relative entropy of the detrended weight matrices and we can still think of the fittings as representing the trend of deviation from the steady state distribution of the weights for each layer, but a more concrete interpretation from an information theoretic perspective is still elusive. Trying to tie these results back to the most basic definition of entropy and the equivalent understanding of it being some measure of surprise is non-obvious and we may only consider these results as a metric in and of themselves without a more theoretic realization beyond simple information change. Perhaps for an easier understanding we can simply measure the entropy of the detrended distributions individually, but this refocusing is something we learned in hindsight and was not a more obvious direction of investigation at the beginning.

4.1 The Good

The results of this thesis are very useful in that we have arrived at some alternative metrics for quantifying good training of DLDMD. Both Figures 3.2 and 3.9 give us an idea of which dimension is optimal via the average of the slopes of all the fittings of the layers and we can surmise that smaller average slopes and standard deviations tend to correspond to better training and comparisons with the phase space results and loss curves justify this claim. At the very least, we see that differences are apparent in our metrics for hyperparameters that we know to be successful versus those that are not and it may be useful to incorporate such analysis into a model to decide whether training should end at any given stage in favor of different parameters. The most appropriate way to integrate this into a training scheme will be difficult, but it should be possible to run this in parallel to a training scheme by saving sets of weights from previous epochs and choosing slope bounds that you would like the averages to fall into. As a proof of concept, we have found that the data in the form of the weights of a model can be used to characterize the model's evolution outside of the traditional metrics of ML and gives us a statistical viewpoint of the seemingly random walk that these models take during training. We have also shown how individual layers of a given

model can have an outsized impact on the effectiveness of the model. The egde layers evolved in a very different way than the hidden layer and Figures 3.5 and 3.12 show this.

4.2 Optimal Parameters

While we believe that we have struck upon some useful results with respect to finding different behavior and trends in some of the different values we have computed, it must be made clear that some assumptions were made on our part that might not have been valid ones to makes given what we have learned since the outset. Perhaps the biggest assumption made is that we have sets of optimal parameters at all for the DLDMD algorithm. There has not been much study into this algorithm yet, and the only measure of its validity is given by [1] in their results and our replications which admittedly rely on the “eye-ball” metric. Predicting future states in multidimensional nonlinear dynamical systems from time-series data is very tricky and it is difficult to quantify accuracy or whether key phase space features are respected in the predictions. Dealing with separatrices and the concentric centers, the prediction results, and the activation function issues of Duffing in particular casts some doubt on the parameters we have taken to be optimal.

4.3 Non-unitary densities

When creating the empirical densities in Chapter 3, often we end up deriving density functions that don’t exhibit a core property of density functions: The sum of their probabilities does not integrate to 1. KDE was doubtless designed for less pathological data than the kind we seem to be extracting from the weight matrices of the model. This perhaps conflicts with the assumption that we should be able to extract a density from the weight matrices at all. After some investigation, we have come to realize that this is a consequence of the bandwidth use in KDE. For our density estimation, we made no assumptions about the shape or modality of the data and so the existing literature led us to using the ISJ to find an optimal bandwidth for each probability density. The resulting densities are full of narrow peak and wide troughs, reminiscent of Dirac’s comb. The bandwidths due to ISJ were on the order of 10^{-6} and we found that using larger bandwidths led to overall smoother densities that we were able to numerically verify have approximately unitary area. The varying shapes based on bandwidth are another indicator of the inherent tradeoff between bias and variance in any model.

We have no reason to believe that there is anything at all wrong with the software package being used for KDE and instead postulate that a more sophisticated scheme of integration, one that does not rely as much on smooth data, is needed for the

optimal bandwidth parameter. One silver-lining we can deduce from this is that the apparent failure of boiler plate numerical integration can give us an idea of how much variance a given derived density has and we can use that make other inferences about the model itself.

4.4 Statistical issues with edge layers

The weight matrices are used in the model for linear transformations; the hidden layers have approximately 128^2 entries while the edge layers have about $128 \cdot N_S$ or $128 \cdot N_O$ entries, due to each hidden layer having 128 neurons and N_S and N_O are the numbers of physical dimensions of the systems and the latent dimension, respectively. In either case, the number of neurons is much greater than either N_S or N_O so from a statistical standpoint the edge layers don't have as much data for the analysis. As we've elaborated on in Chapter 3, we do notice marginally different behavior in the results for the edge layers than we do for the hidden layers and it's as of yet unclear whether this is evidence of the validity of our hypothesis or a consequence of having less data to fully characterize their evolution. For the range of latent dimensions we ran the model with, the hidden layers have approximately 16000 data points and the edge layers can have as few as 258 data points or as many 1161 data points. This spread alone leads to a belief that the higher latent dimensions should yield better results because there are more trainable parameters and it is consistent with the expectation that more observables leads to an improved EDMD in the lifted coordinates. It is, however, not consistent with all of the results from DLDMD or the analysis in this thesis.

4.5 Future work

In the interest of future exploration of the ideas in this thesis, another ML model that would be excellent to examine would be a classifier of some kind. Such a problem would also allow us to use alternative metrics such as model accuracy (which is much easier to quantify than the for the dynamical systems we investigate) along with our analysis to get a better idea of the limits of this discovery of good training through the frame of information change. Such classifiers are all over the modern literature of ML and would have great interest in a method for determining good training during training would be of immense utility. Naturally, image processing applications of ML would be worth looking at through this lense considering how much more complex images are as inputs and that most applications implicitly involve examining fine details with respect to the amount of information the image contains. It would also provide an opportunity to experiment with NN architectures other than the Sequential model of DLDMD. Change in information content might also be a meaningful

consideration for unsupervised training in that we have another way to quantify if the model is arriving at a configuration that accomplishes the given task or not when metrics are not available due to lack of training targets.

BIBLIOGRAPHY

- [1] D. ALFORD-LAGO, C. W. CURTIS, A. T. IHLER, AND O. ISSAN, *Deep Learning Enhanced Dynamic Mode Decomposition*, Chaos: An Interdisciplinary Journal of Nonlinear Science, 32 (2022).
- [2] Z. I. BOTEV, J. F. GROTOWSKI, AND D. P. KROESE, *Kernel density estimation via diffusion*, The Annals of Statistics, 38 (2010), pp. 2916 – 2957.
- [3] S. BRUNTON, B. R. NOACK, AND P. KOUMOUTSAKOS, *Machine learning for fluid mechanics*, Ann. Rev. Fluid Mech., 52 (2020), pp. 477–508.
- [4] V. A. EPANECHNIKOV, *Non-parametric estimation of a multivariate probability density*, Theory of Probability & Its Applications, 14 (1969), pp. 153–158.
- [5] A. GERON, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, O'Reilly Media, Inc., 2nd ed., 2019.
- [6] M. HOLLANDER, D. WOLFE, AND E. CHICKEN, *Nonparametric Statistical Methods*, Wiley Series in Probability and Statistics, Wiley, 2013.
- [7] K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, (1989).
- [8] B. KOOPMAN, *Hamiltonian systems and transformations in Hilbert space*, Proc. Nat. Acad. Sci., 17 (1931), pp. 315–318.
- [9] M. A. KRAMER, *Nonlinear principal component analysis using autoassociative neural networks*, Aiche Journal, 37 (1991), pp. 233–243.
- [10] S. KULLBACK AND R. A. LEIBLER, *On Information and Sufficiency*, The Annals of Mathematical Statistics, 22 (1951), pp. 79 – 86.
- [11] B. LUSCH, J. N. KUTZ, AND S. L. BRUNTON, *Deep learning for universal linear embeddings of nonlinear dynamics*, Nature Comm., 9 (2018), p. 4950.
- [12] E. PARZEN, *On Estimation of a Probability Density Function and Mode*, The Annals of Mathematical Statistics, 33 (1962), pp. 1065 – 1076.
- [13] M. ROSENBLATT, *Remarks on Some Nonparametric Estimates of a Density Function*, The Annals of Mathematical Statistics, 27 (1956), pp. 832 – 837.
- [14] D. E. RUMELHART, J. L. MCCLELLAND, AND C. PDP RESEARCH GROUP, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, Cambridge, MA, USA, 1986.
- [15] P. SCHMID, *Dynamic mode decomposition of numerical and experimental data*, J. Fluid Mech., 656 (2010), pp. 5–28.
- [16] D. W. SCOTT AND G. R. TERRELL, *Biased and unbiased cross-validation in*

- density estimation*, Journal of the American Statistical Association, 82 (1987), pp. 1131–1146.
- [17] C. E. SHANNON, *A mathematical theory of communication*, The Bell System Technical Journal, 27 (1948), pp. 379–423.
 - [18] S. H. STROGATZ, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*, Westview Press, 2000.
 - [19] S. THEODORIDIS, *Machine Learning: A Bayesian and Optimization Perspective*, Academic Press, Inc., USA, 1st ed., 2015.
 - [20] M. WILLIAMS, I. G. KEVREKIDIS, AND C. W. ROWLEY, *A data-driven approximation of the Koopman operator: extending dynamic mode decomposition*, J. Nonlin. Sci., 25 (2015), pp. 1307–1346.