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

July 15, 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

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

Write this once everything else is written.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

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.

Say more once you've thought of it.

1.1   What is E-DMD?

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

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

As established in the introduction, there is an old and venerable literature dedicated to deriving system 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 $\boldsymbol{\varphi}(t; \boldsymbol{x}) = \boldsymbol{y}(t)$ to be the flow map affiliated with the initial condition $\boldsymbol{x}$ and denote $g : \mathcal{M} \to \mathbb{C}$ to be a square integrable observable of the system then, according to [7], there exists a linear representation of the flow map given by $\mathcal{K}$;

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

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've 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 $\{\phi_\ell\}$, of the Koopman Operator such that

$$\mathcal{K}^t \phi_\ell = \exp(t\lambda_\ell)\phi_\ell \implies g(\boldsymbol{x}) = \sum_{\ell \in \mathbb{N}} c_\ell \phi_\ell(\boldsymbol{x}),$$

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

$$\mathcal{K}^t g(\boldsymbol{x}) = \sum_{\ell \in \mathbb{N}} a_\ell \phi_\ell(\boldsymbol{\varphi}(t, \boldsymbol{x}))$$

The most useful property of this framing is that we have a global linearization of the flow, with a major caveat: Generally, finding these eigenvalues and eigenfunctions is impossible. This led to the development of the already mentioned and much much lauded Dynamic Mode Decomposition (DMD) and it's extensions, which seeks to numerically approximate a finite number of these modes. We'll focus on the particular extension that [1] implements: Extended DMD (E-DMD) [10]. With E-DMD, we suppose that

$$g(\boldsymbol{x}) = \sum_{\ell=1}^{N_O} a_\ell \psi_\ell(\boldsymbol{x})$$

which is to say that the observable g exists in a finite dimensional subspace of $L^2(\mathcal{M})$, applying the Koopman operator implies that

$$\mathcal{K}^{\delta t} g(\boldsymbol{x}) = \sum_{\ell=1}^{N_O} a_\ell \phi_\ell(\boldsymbol{\varphi}(\delta t, \boldsymbol{x}))$$

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

for discrete time step δt. K_O is the $N_O \times N_O$ matrix that minimizes

$$\boldsymbol{K}_O = \operatorname*{argmin}_{K} ||\Psi_+ - K\Psi_-||_F^2$$

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

$$\Psi_- = \left(\Psi_1 \ \Psi_2 \ \cdots \Psi_{N_T}\right), \quad \Psi_+ = \left(\Psi_2 \ \Psi_3 \ \cdots \Psi_{N_T+1}\right)$$

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

$$\Psi_- = U\Sigma W^\dagger \implies \boldsymbol{K}_O = \Psi_+ W \Sigma^{-P} U^\dagger$$

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:

$$E_r(\boldsymbol{K}_O) = \left|\left|\Psi_+(I - WW^\dagger)\right|\right|_F$$

Finding the eigenvalues, eigenfunctions, and Koopman modes comes down to an eigen-decomposition, from which the dynamics can be approximated as

$$y(t; \boldsymbol{x}) \approx V \exp(t\Lambda) V^{-1} \boldsymbol{\Psi}(\boldsymbol{x})$$

where $\boldsymbol{K}_O = VTV^{-1}$, $\Lambda_{\ell\ell} = \ln(T_{\ell\ell})/\delta t$ is a diagonal matrix and $\boldsymbol{\Psi}$ is the representation of the initial condition in terms of the observables.

Expand on everything else above for a more complete picture of E-DMD.

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 researcher 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.

Expand once you've got more to say.

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 Neural Networks (NNs). In 1989, [6] 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.

From this we can consider ML as a tool of regression. 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

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

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

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

where

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

and

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

With this framing, ML methods can be cast as attempts to find an estimator relative to a given training 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[5] 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 it's limitations

Incomplete picture, add more detail later. Consider re-skimming Jay's paper.

The key innovation of [1] is to use a neural network to come up with the collection of observables on $\{\boldsymbol{y}_j\}$ that allow for the best prediction of future system states. This is implemented by defining an encoder $\mathcal{E} : N_S \to N_O$ and decoder $\mathcal{D} : N_O \to N_S$ composed of Dense layers such that

$$(\mathcal{D} \circ \mathcal{E})(\boldsymbol{x}) = \boldsymbol{x}$$

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

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{recon}} + \alpha_2 \mathcal{L}_{\text{dmd}} + \alpha_3 \mathcal{L}_{\text{pred}} + \alpha_4 \left|\left| \boldsymbol{W}_g \right|\right|_2^2 \tag{1.1}$$

where

$$\mathcal{L}_{\text{recon}} = \frac{1}{N_T + 1} \sum_{j=1}^{N_T+1} \left|\left| \boldsymbol{y}_j - (\mathcal{D} \circ \mathcal{E})(\boldsymbol{y}_j) \right|\right|_2, \tag{1.2}$$

$$\mathcal{L}_{\text{dmd}} = E_r(\boldsymbol{K}_O), \tag{1.3}$$

$$\mathcal{L}_{\text{pred}} = \frac{1}{N_T} \sum_{j=1}^{N_T} \left|\left| \boldsymbol{y}_{j+1} - \mathcal{D}\left(V T^j V^{-1} \mathcal{E}(x) \right) \right|\right|_2, \tag{1.4}$$

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. $\mathcal{L}_{\text{DMD}}$ 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 observable; 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. $\boldsymbol{W}_g$ 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 4 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 E-DMD with Dense NNs and some meaningful choices of loss function as an innovation was made by [1, 3, 8], 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 into the addition of gaussian noise led to the discovery of the machine being quite robust to data due to the characteristics of it's loss function and additional noise mitigation was accomplished by implementing convolutional layers instead of traditional dense layers.

1.4   Statement of Dilemma

Having estabilshed 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 $\boldsymbol{x} \in \mathbb{R}^d$ through a layer L can be written as

$$L(\boldsymbol{x}; A, \sigma, b) = \sigma(A\boldsymbol{x} + b)$$

for some matrix $A \in \mathbb{R}^{D \times d}$, vector $b \in \mathbb{R}^D$ and (typically nonlinear) activation function $\sigma : \mathbb{R}^D \to \mathbb{R}^D$. For a given layer, σ is a fixed hyperparameter of the machine, but both A and b have entries that are tuned by the optimizer as training takes places. Training takes place on a per epoch basis for most models, 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

$$Q_{n+1} = P(Q_n) \quad \text{(Come up with better symbols later)}$$

where P is the training procedure and Q_n represents the configuration of the network's weights at each epoch n. Typically, the initial state Q_0 is "random" in the sense that the weights are selected from a probability distribution. In the interest of examining convergences, one might consider the following limit

$$\lim ||Q_{n+1} - Q_n||_2$$

for some L^2 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. By the noted stochastic nature of the evolution, we consider the entries of A and 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, it is often useful to 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're much more likely to get something close to μ than we are to get something far away from μ. For example; 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 of any given probability distribution. A measure of the uncertainty was proposed in 1948 and this measure is known as *informational entropy* [9] or, simply, entropy. For a continuous distribution with density function $f(x)$, the entropy is given by

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

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

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

and the entropy integral for this evaluates to

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

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 interpretted 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.

When training an ML model, the expectation is that the machine will converge to some configuration that allows it to accomplish the task it was designed for. For the sake of
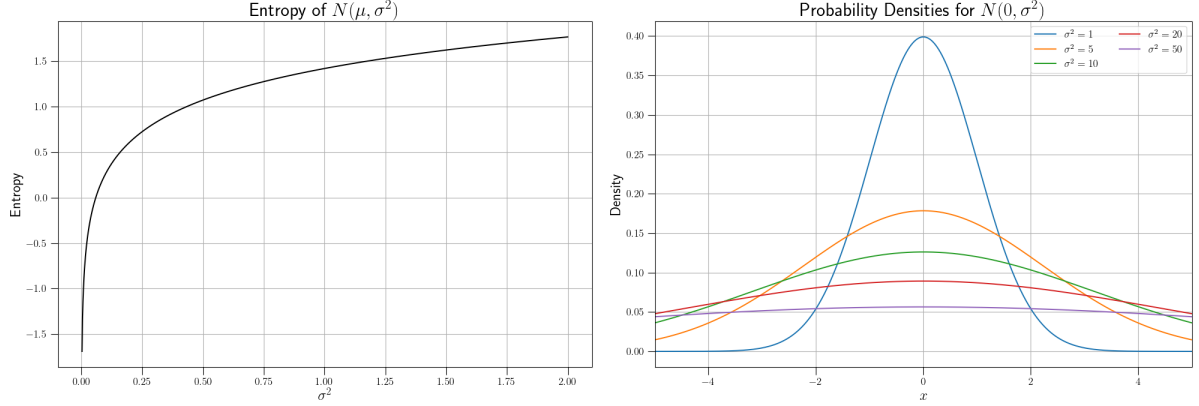
Figure 2.1. Entropy and density function change with respect to σ^2. The base of the logarithm shown here is e, but the base itself is irrelevant due to the shared properties of all logarithms: that they increase monotonically and unbounded. As the variance increases we see higher entropy in the left plot and a flattening of the probability density in the right plot.

2.1 Basic Definitions

The KL Divergence (KLD) is a statistical distance between a pair of probability distributions which measures how different a distribution P is from a 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." If P and Q are discrete distributions defined on the probability space $\mathcal{X}$, then the KLD of P with respect to Q is given by

$$D_{KL}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(P(x)/Q(x) \right) \tag{2.4}$$

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 more standard metrics, it is not symmetric and does not satisfy the triangle equality. In order to deal with something that is, at least, symmetric we'll consider using the symmetric KL Divergence (SKLD) which is defined as

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

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

$$D_{KL}(P \parallel Q) = \int_{\mathcal{X}} p(x) \log \left(p(x)/q(x) \right) \, dx \tag{2.6}$$

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

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

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

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

Figure 2.2 illustrates the KLD between the example normal distributions.
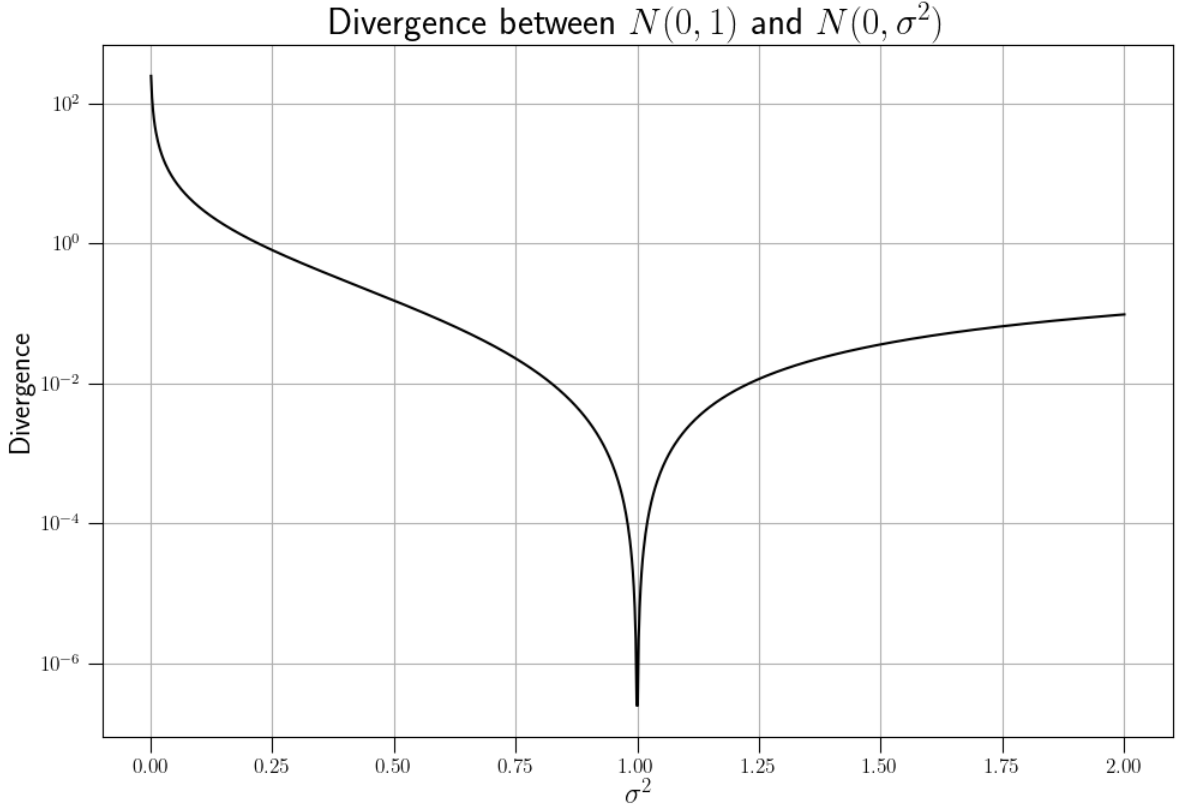
Figure 2.2. KL Divergence example with Normal distributions. As expected, the divergence decreases to 0 as $\sigma^2 \to 1$ from either direction.

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'll be using Kernel Density Estimation (KDE). The most basic idea behind KDE is using a different 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:

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

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 [4], the choice of kernel does not provide much statistical significance; but the choice of the bandwidth parameter h is very crucial for finding a density estimate that approximates the underlying density function appropriately. 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 ...

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 affliated 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

$$\mathbf{W}_{l,\mathcal{E}}^{(n)}, \mathbf{W}_{l,\mathcal{D}}^{(n)} \to \mathbf{W}_{l,\mathcal{E}}^{*}, \mathbf{W}_{l,\mathcal{D}}^{*} \text{ as } n \to \infty \tag{2.9}$$

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

$$\mathbf{W}_{l,\mathcal{E}}^{(n)} = \mathbf{W}_{l,\mathcal{E}}^{*} + \mathbf{W}_{l,\mathcal{E}}^{(n),f}, \ \ \mathbf{W}_{l,\mathcal{D}}^{(n)} = \mathbf{W}_{l,\mathcal{D}}^{*} + \mathbf{W}_{l,\mathcal{D}}^{(n),f}, \tag{2.10}$$

where the fluctuation matrices $\mathbf{W}_{l,\mathcal{E}}^{(n),f}$ and $\mathbf{W}_{l,\mathcal{D}}^{(n),f}$ are defined implicitly. These fluctuations are what we're most interested in studying. Using first-order differencing, we can study the affliated detrended matrices $\delta\mathbf{W}_{l,\mathcal{E}}^{(n)}$ where

$$\begin{aligned} \delta\mathbf{W}_{l,\mathcal{E}}^{(n)} &= \mathbf{W}_{l,\mathcal{E}}^{(n+1)} - \mathbf{W}_{l,\mathcal{E}}^{(n)} \\ &= \mathbf{W}_{l,\mathcal{E}}^{(n+1),f} - \mathbf{W}_{l,\mathcal{E}}^{(n),f} \end{aligned}$$

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 affliated empirical probability distribution $p_{l,\mathcal{E}}^{(n)}(w)$. With these distributions we know find the KLD between consecutive distributions to generate the following sequence:

$$D = \left\{ D_{KL}\left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E-2} \tag{2.11}$$

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 will 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

$$\mathbf{W}_{\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_l}, \mathbf{W}_{\mathcal{D}} = \left\{ \mathbf{W}_{l,\mathcal{D}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_l}, \tag{2.12}$$

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 and only differ by the dimension of their inputs and outputs. The following procedures are identical for both sets, so we'll only be referencing the encoder set from here on out. For each layer l, the set of detrended matrices are computed

$$\delta\mathbf{W}_{l,\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n+1),f} - \mathbf{W}_{l,\mathcal{E}}^{(n),f} \right\}_{n=1}^{N_E-1}, \tag{2.13}$$

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/, make this a URL later**), which implements several algorithms for KDE in python including the bandwidth computation of ISJ. The kernel used is the Epanechnikov kernel, defined as

$$K(x) = \begin{cases} \frac{3(1-x^2)}{4} & |x| < 1 \\ 0 & \text{otherwise} \end{cases}, \tag{2.14}$$

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

$$p_{l,\mathcal{E}} = \text{KDE}(\delta\mathbf{W}_{l,\mathcal{E}}) = \left\{ p_{l,\mathcal{E}}^{(n)}(w) \right\}_{n=1}^{N_E-1} \tag{2.15}$$

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 10001 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're truly interested in: the relative entropy of consecutive densities and the associated change in information:

$$D_l = \left\{ D_{SKL}\left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E-2} \tag{2.16}$$

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 asymmertric 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 have the same support and use the 10001 discretized points with Simpson's rule 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 2.3 shows an example of the divergence over the whole training.

　　We're interested in what these divergences tell us about information flow in the model's weights, and as such we seek a fitting of the data. Any overall trends are 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 trens, so for the time being we ignore it. In the interest of trying to identify a classifier of good training, we then find the average and variance of the slopes of the linear fits of the layers' 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.

Figure 2.3. Example of divergences for layer Dec 2 of Van der Pol model with latent dimension 8 with linear fitting and confidence bands.

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 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 $N_O = 15$, the algorithm detailed in Chapter 2 was implemented for the following systems: The Duffing and Van der Pol oscillators and (maybe) Lorenz-63. The initial hope of our analysis is the data points that our algorithm generates will show clear trends that will allow us to easily differentiate the optimal parameter choice for the model from suboptimal or even bad parameter choices. Given the model's task of learning an imbedding 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 certaintly accomplish something as the model trains, but the dimension changes that help E-DMD 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

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

There are many sets of coefficients for which many types of behvior are exhibited by the system; but for our purposes we'll only consider the following configuration:

$$\ddot{x} - x + x^3 = 0 \tag{3.2}$$

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

$$\begin{cases} \dot{x} = y \\ \dot{y} = x - x^3 \end{cases} \tag{3.3}$$

The interesting attributes of this system are that it has 3 fixed points, 2 of which are non-linear centers with the third being a saddle point with homoclinic
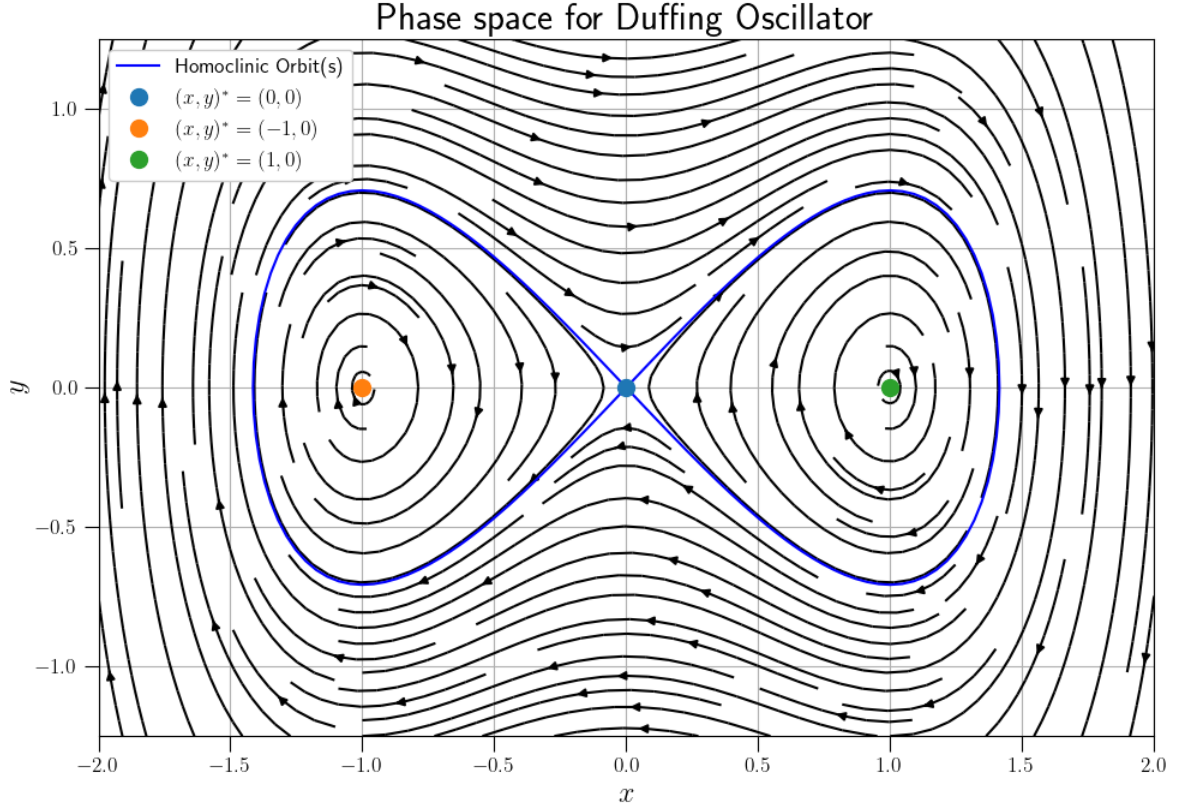
Figure 3.1. Duffing phase space.

connections that encompass the other 2 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. 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 mention 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 3. For reasons still under investigation, the RELU activation function causes the Singular Value Decomposition to fail if the latent dimension is 3 or greater and the algorithm subsequently ends with

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

Hyperparameter	Value
Initial Conditions	10000
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}

an error. After experimentation, it was found that using ELU instead circumvents this problem without significantly affecting model training.

Having established this, we'll 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 the for latent dimensions 2 through 5, the average slope is small in magnitude with a small to medium standard deviation relative to the scale with latent dimension 6 showing not only a much larger average slope, but as also much more variance. We may contrast this with the losses in Figure 3.3 for each latent dimension, which seems to show that latent dimension 5 should be the optimal choice with all other choices being more or less equivalent. This does not match the results of the phase space representation, which show roughly that dimension 3 is best. If we examine Figure 3.4, which shows the average bandwidth from the ISJ algorithm across all layer for a given latent dimension per epoch, we can see that 3 has a slightly different trend and separates cleanly from each of the other choices. While this behavior tracks with our understanding that this dimension is best and should be easily differentiated from the rest, it is inconsistent the fact that the other bandwidth curves are banded together
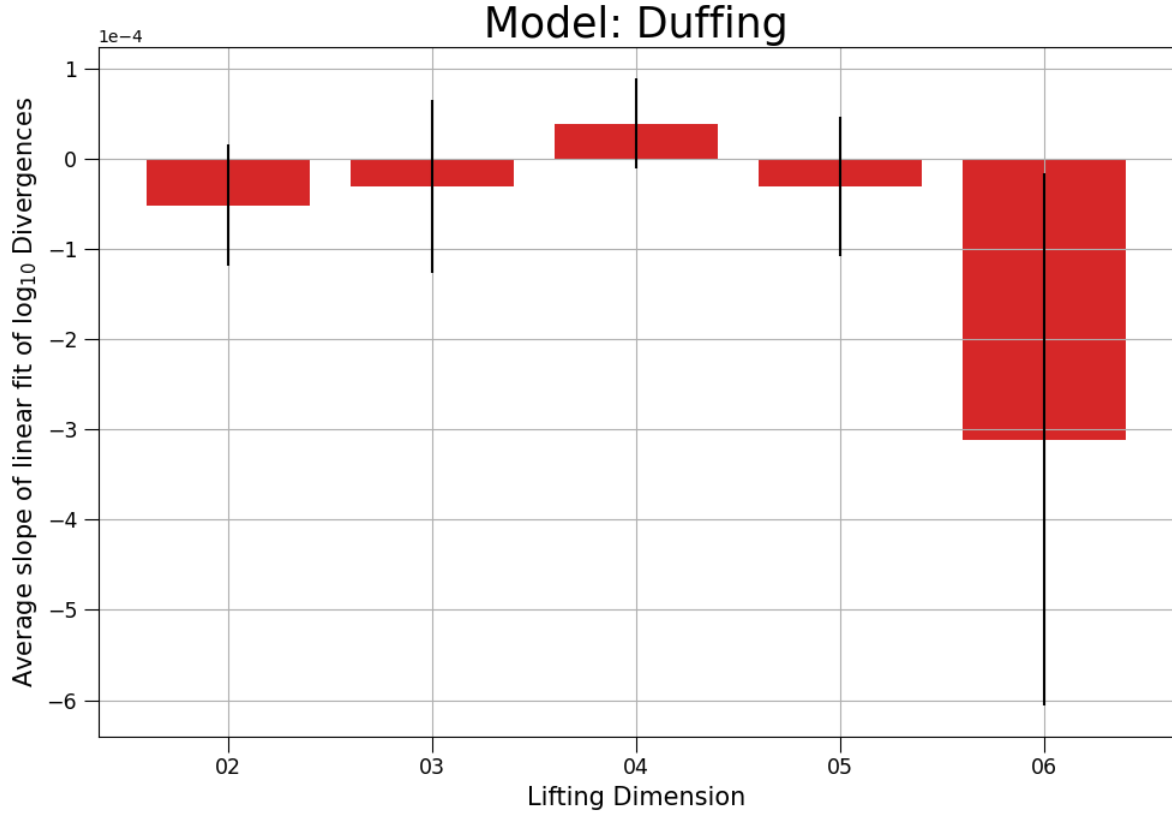
Figure 3.2. Duffing average slopes with standard deviations.

but don't have much in common with respect to Figures 3.2 and 3.3 for both the fitted slopes and loss curves.

For a more granular view of the individual fittings for each layer of the model, we have Figures 3.5 and 3.6. 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 have a information change trend that is further from the theoretical steady-state than for the other layers of the model.

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

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

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

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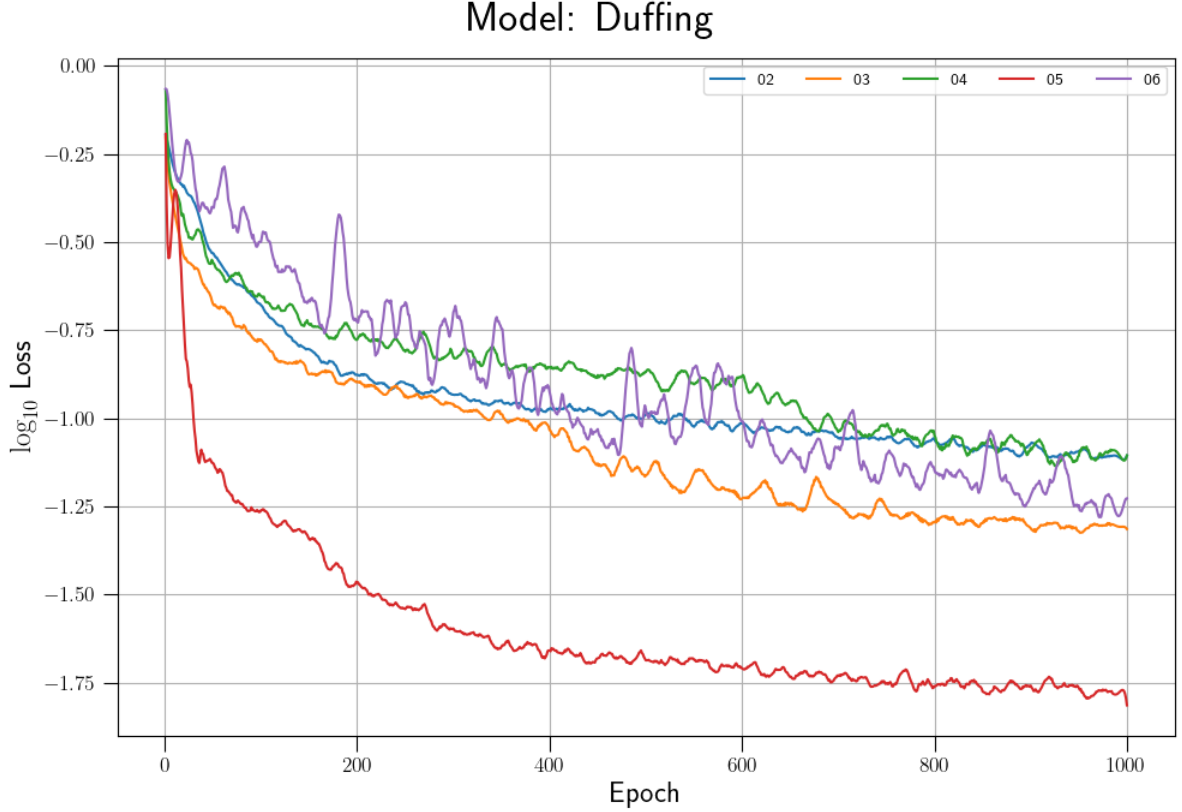
Figure 3.3. Duffing loss plots, smoothed to accentuate trends.

order equations is

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

Again, by introducing $y = \dot{x}$. 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 3.7 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 in the limit cycle; but since all orbits approach the limit cycle there should be much variability with respect to initial conditions. 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 Rectified Linear Unit. This difference is minor enough that a Van der Pol hyperparameters table is omitted.
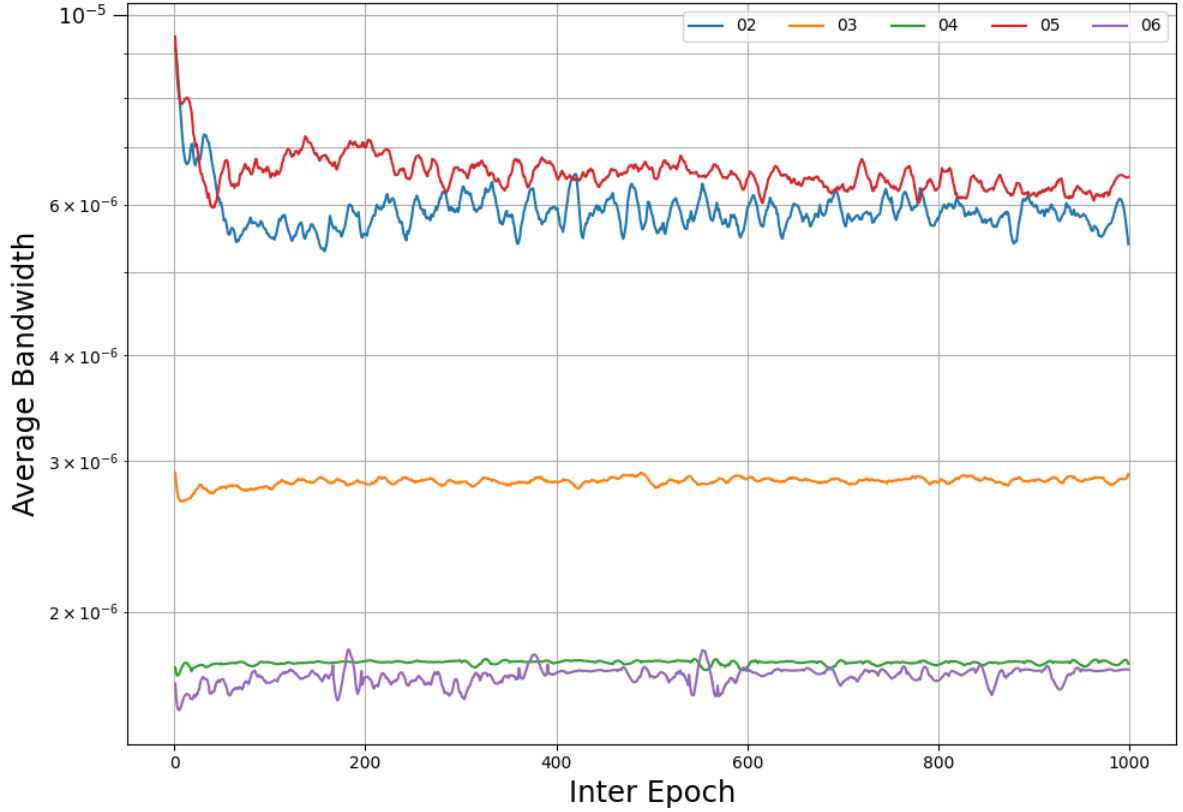
Figure 3.4. Duffing Average bandwidth.

Figure 3.8, 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 larger average slopes and more variety with respect to the standard deviations for each dimension. Based on the phase plane results, we should expect 8 to correspond to the best training so it seems that a small in magnitude slope is indicative of good training and this in consistent for some of the loss curves in Figure 3.9. The slope of 5 is essentially 0 and we that it has a loss curve almost indistinguishable from that of 8; and we can make a similar observation about 7 as well. However, 10 also has a small slope but the loss curve is markedly different from the optimal choices and seems to match that of 3 which has a very large slope comparitively. One quality the optimal trainings seem to have in common is that they have smaller standard deviations than the trainings with worse loss curves.
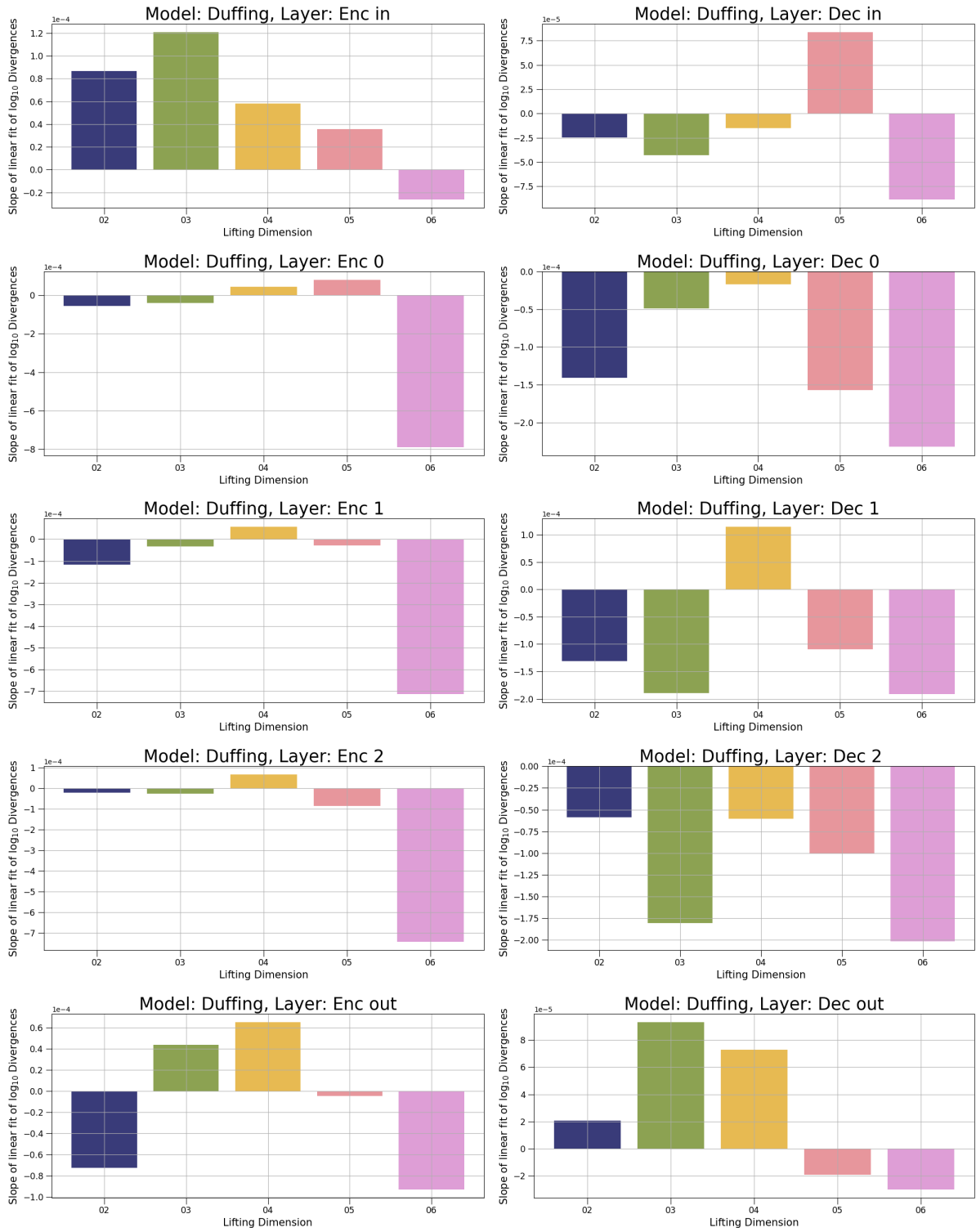
3.3   Lorenz (Maybe)
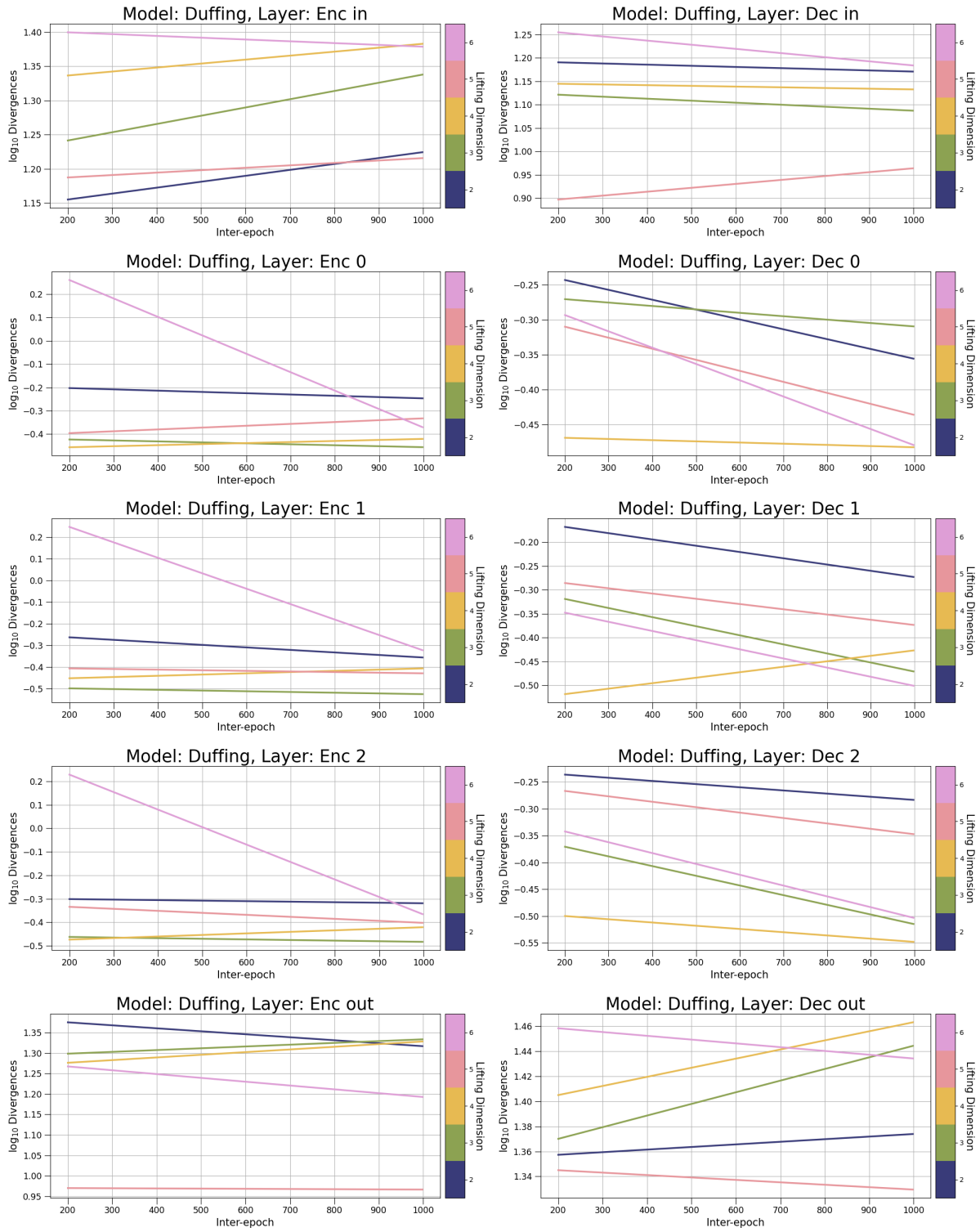
Figure 3.5. Duffing slope linear fits by layer.
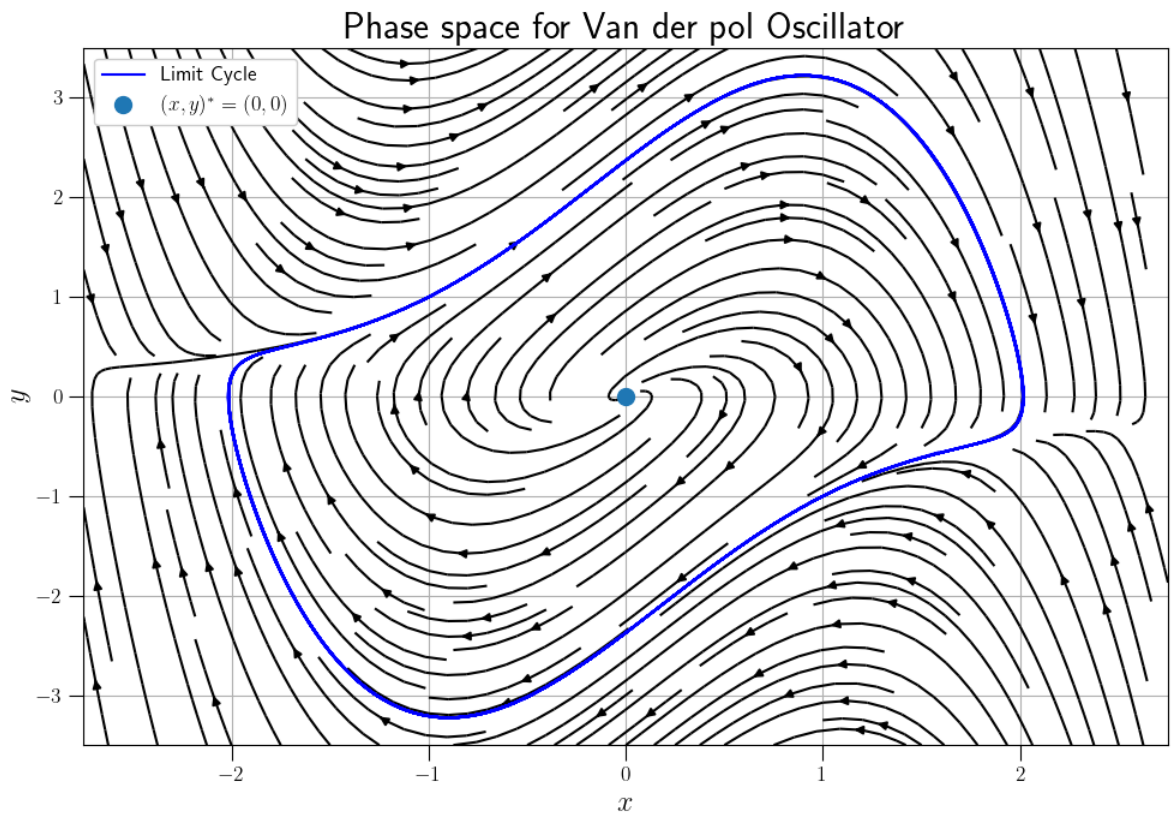
Figure 3.6. Duffing slope linear fits by layer.

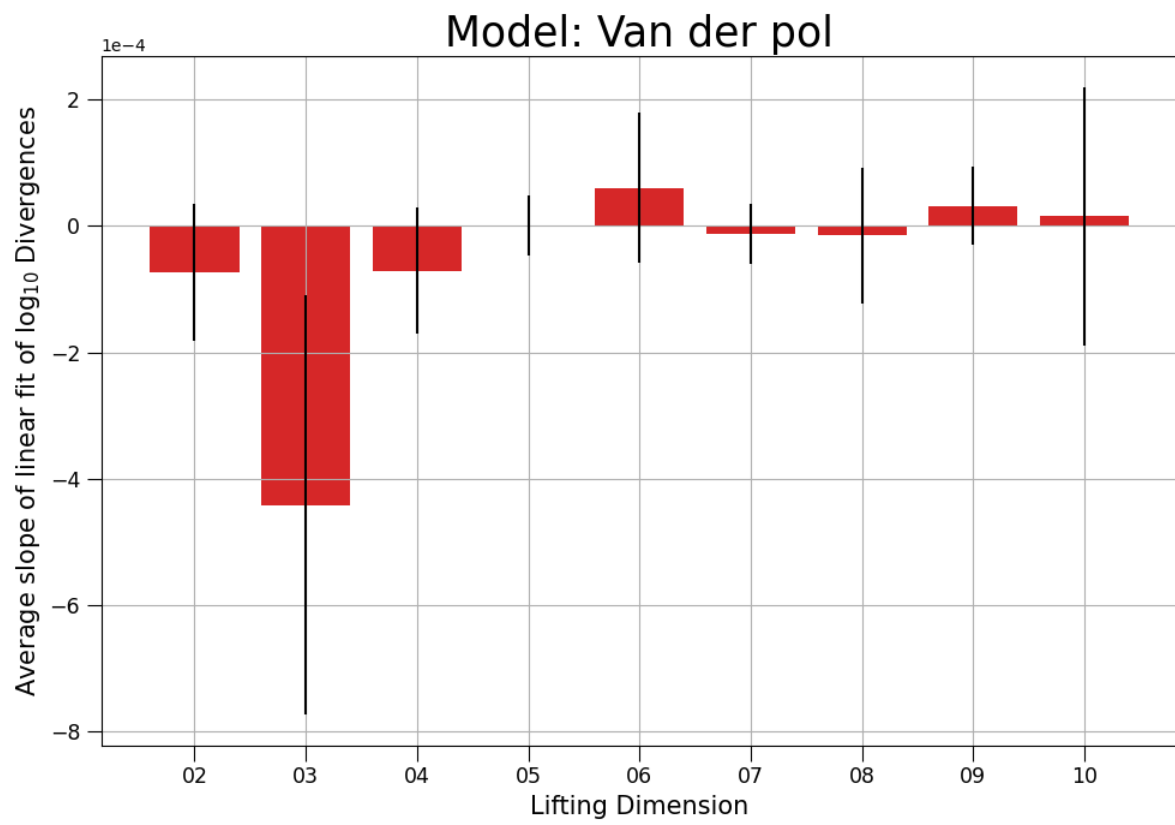**Figure 3.7. Van der Pol phase space.**

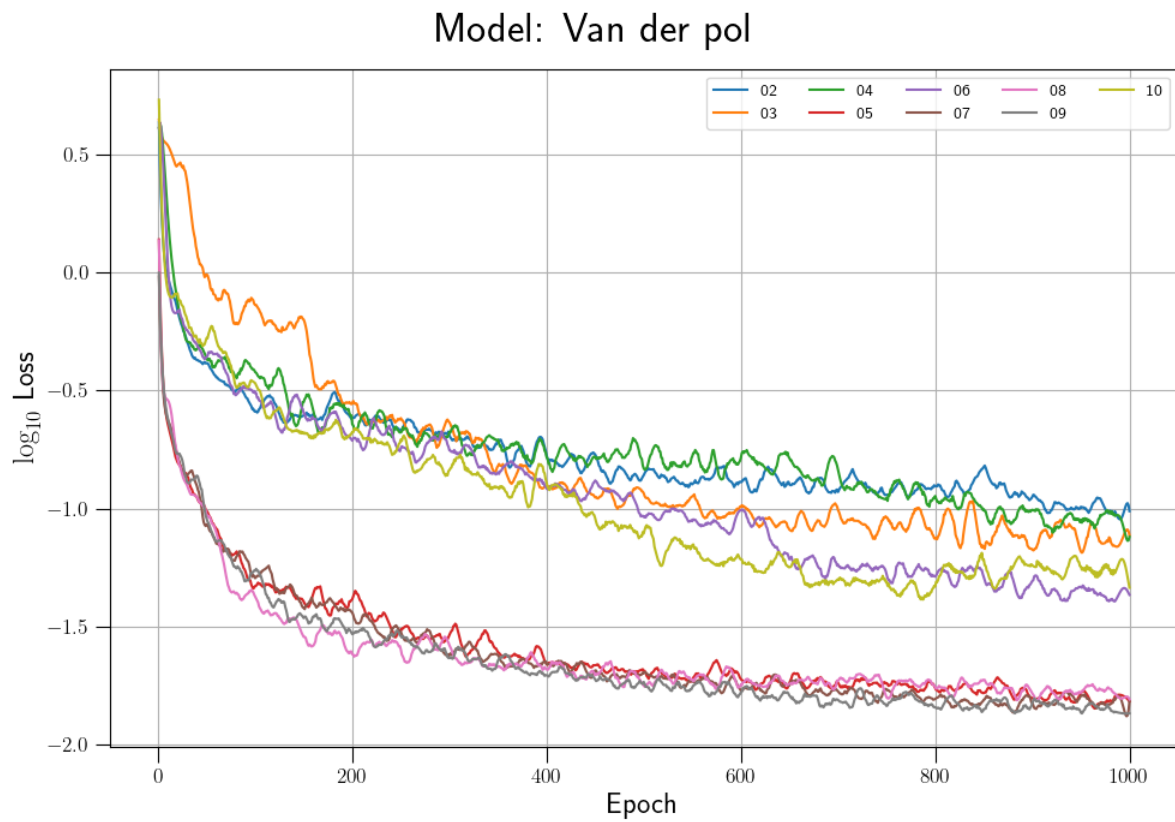**Figure 3.8. Van der Pol average slopes with standard deviations.**

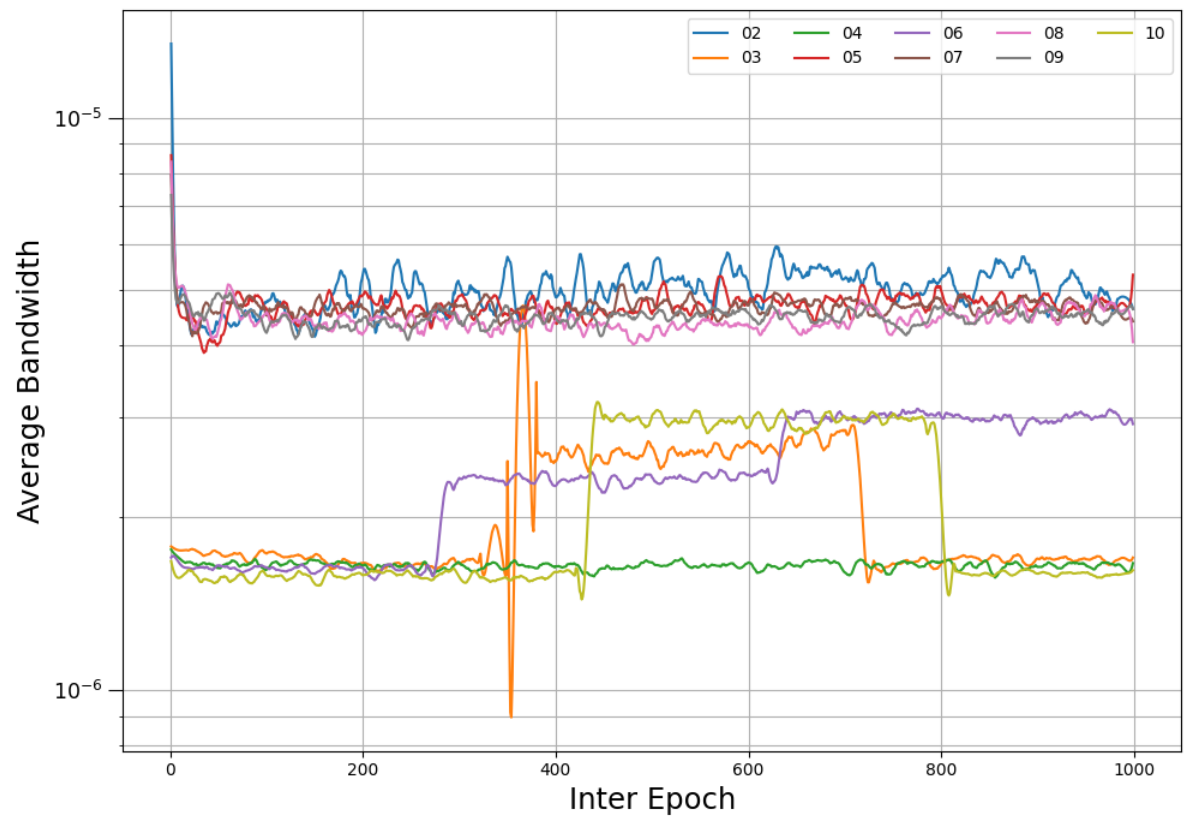**Figure 3.9. Van der Pol loss plots, smoothed to accentuate trend.**

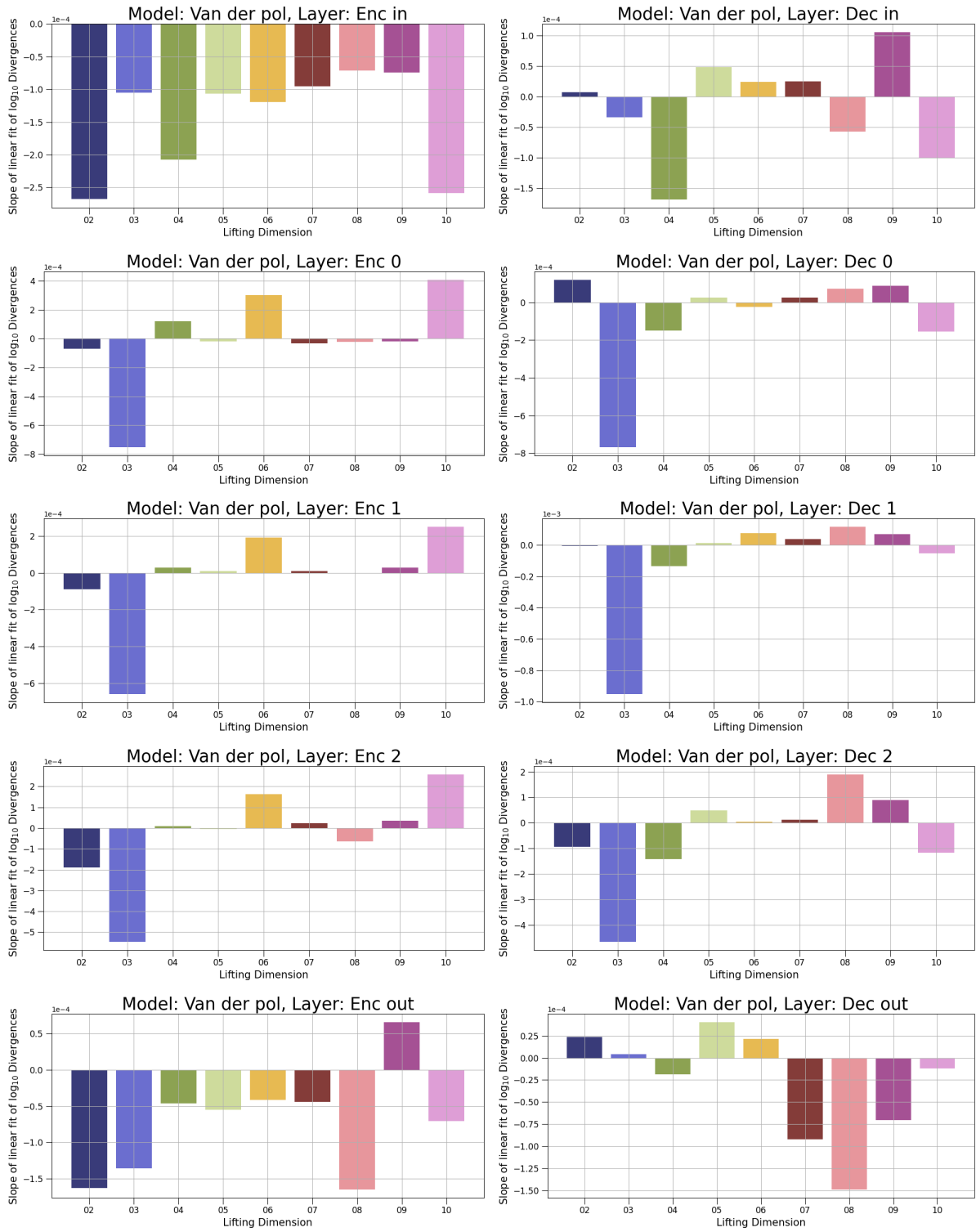**Figure 3.10. Van der Pol Average bandwidth.**

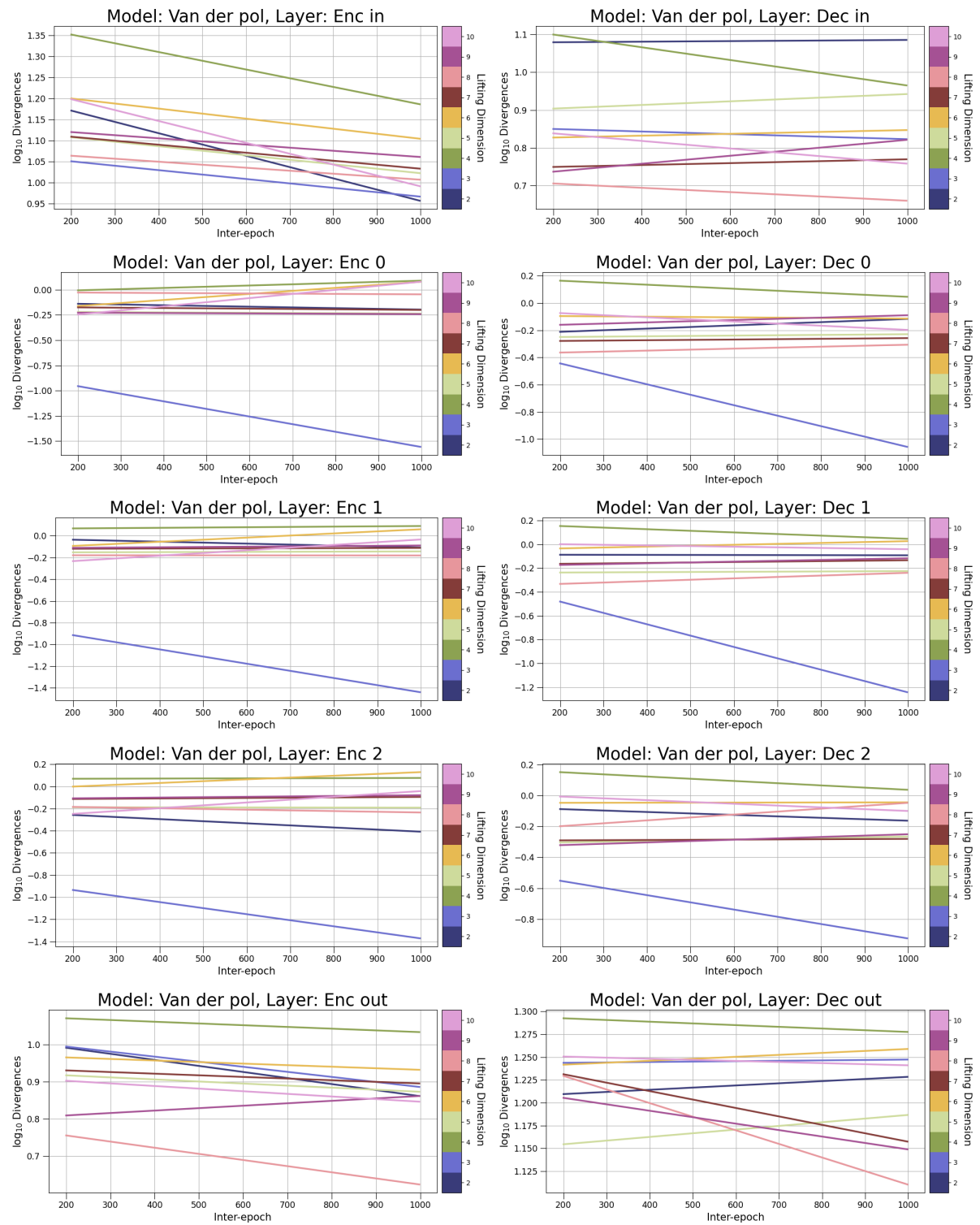**Figure 3.11.** Van der Pol slope linear fits by layer.

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

CHAPTER 4

DISCUSSION

Mention potential issues with results.

Mention issues with KDE not giving a "unitary" density. When creating the empirical densities in Chapter 3, often we end up deriving density functions that don't 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. This issue was rectified (though, you might say subverted) by numerically integrating the non-unitary densities and dividing by their area to force them into having the desired area. Due to their narrow support, this introduces a new issue in that some of the densities now have individual weights with density greater than 1.

Mention misgiving about symmetric KLD.

BIBLIOGRAPHY

[1] D. ALFORD-LAGO, C. W. CURTIS, A. T. IHLER, AND O. ISSAN, *Deep Learning Enhanced Dynamic Mode Decomposition*, (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] M. HOLLANDER, D. WOLFE, AND E. CHICKEN, *Nonparametric Statistical Methods*, Wiley Series in Probability and Statistics, Wiley, 2013.

[6] K. HORNIK, M. STINCHCOMBE, AND H. WHITE, *Multilayer Feedforward Networks are Universal Approximators*, Neural Networks, (1989).

[7] B. KOOPMAN, *Hamiltonian systems and transformations in Hilbert space*, Proc. Nat. Acad. Sci., 17 (1931), pp. 315–318.

[8] B. LUSCH, J. N. KUTZ, AND S. L. BRUNTON, *Deep learning for universal linear embeddings of nonlinear dynamics*, Nature Comm., 9 (2018), p. 4950.

[9] C. E. SHANNON, *A mathematical theory of communication*, The Bell System Technical Journal, 27 (1948), pp. 379–423.

[10] 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.