A study on quantifying effective training of DLDMD

Joseph A.G. Diaz

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

August 1, 2022

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. The data-driven method Dynamic Mode Decomposition and it's extensions offer a compelling avenue in the problem of prediction from time-series data.
The marriage of these methods with Machine Learning and Neural Networks allows for the leveraging the power of these tools in the space.
If standard metrics of model training are unavailable,

Introduction - Koopmanism

We seek a predictive model for a time series $\{\boldsymbol{y}_j\}_{j=1}^{N^T+1}$, which are the measurements of a dynamical system of the form

$$\frac{d\boldsymbol{y}}{dt} = f\left(\boldsymbol{y}(t)\right), \quad \boldsymbol{y}(0) = \boldsymbol{x} \in \mathcal{M} \subseteq \mathbb{R}^{N_s} \tag{1}$$

Denote $\varphi(t;\boldsymbol{x}) = \boldsymbol{y}(t)$ to be the flow map from $\boldsymbol{x}$ and $g : \mathcal{M} \to \mathbb{C}$, be a square integrable observable then, by [5], $\exists \mathcal{K}$ such that

$$\mathcal{K}^t g(\boldsymbol{x}) = g(\varphi(t;\boldsymbol{x})), \tag{2}$$

Finding the eigen-values $\{\lambda_\ell\}$ and eigen-functions $\{\phi_\ell\}$ of $\mathcal{K}$ yields

$$\mathcal{K}^t \phi_\ell = \exp(t\lambda_\ell)\phi_\ell \implies g(\boldsymbol{x}) = \sum_{\ell \in \mathbb{N}} a_\ell \phi_\ell(\boldsymbol{x}), \tag{3}$$

A modal decomposition of g.

From here advancing the dynamics to time t is equivalent to writing

$$\mathcal{K}^t g(\boldsymbol{x}) = \sum_{\ell \in \mathbb{N}} a_\ell \exp(t\lambda_\ell)\phi_\ell(\boldsymbol{x}) \tag{4}$$

This is the basis of Dynamic Mode Decomposition (DMD). If we suppose

$$g(\boldsymbol{x}) = \sum_{\ell=1}^{N_O} a_\ell \psi_\ell(\boldsymbol{x}) \tag{5}$$

with basis $\{\psi_\ell\}_{\ell=1}^{N_O}$ of a subspace, then applying $\mathcal{K}$ for discrete time implies that

$$\mathcal{K}^{\delta t} g(\boldsymbol{x}) = \sum_{\ell=1}^{N_O} a_\ell \exp(\delta t\lambda_\ell)\psi_\ell(\boldsymbol{x}) \tag{6}$$

$$= \sum_{\ell=1}^{N_O} \psi_\ell(\boldsymbol{x})(\boldsymbol{K}_O^T \boldsymbol{a})_\ell + r(\boldsymbol{x}; \boldsymbol{K}_O) \tag{7}$$

Where

$$\boldsymbol{K}_O = \underset{K}{\operatorname{argmin}} ||\boldsymbol{\Psi}_+ - K\boldsymbol{\Psi}_-||_F^2 \tag{8}$$

We define $\boldsymbol{\Psi}_\pm$ to be

$$\boldsymbol{\Psi}_- = (\Psi_1\ \Psi_2\ \cdots \Psi_{N_T}), \quad \boldsymbol{\Psi}_+ = (\Psi_2\ \Psi_3\ \cdots \Psi_{N_T+1}) \tag{9}$$

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 a singular value decomposition (SVD), with which we can write

$$\boldsymbol{\Psi}_- = \mathbf{U}\boldsymbol{\Sigma}\mathbf{W}^\dagger \implies \boldsymbol{K}_O = \boldsymbol{\Psi}_+\mathbf{W}\boldsymbol{\Sigma}^{-P}\mathbf{U}^\dagger \tag{10}$$

Introduction - Flow reconstruction

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

$$\boldsymbol{K}_O = \mathbf{V}\mathbf{T}\mathbf{V}^{-1}, \tag{11}$$

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

$$y(t;\boldsymbol{x}) \approx \boldsymbol{K}_m \exp(t\boldsymbol{\Lambda})\mathbf{V}^{-1}\boldsymbol{\Psi}(\boldsymbol{x}) \tag{12}$$

where $\boldsymbol{\Psi}$ is the representation of the initial condition in terms of the observables, $\boldsymbol{K}_m$ 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

$$\boldsymbol{x} = \sum_{\ell=1}^{N_O} \mathbf{k}_\ell \phi(\boldsymbol{x}), \tag{13}$$

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

Introduction - 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.

Introduction - Function approximation and 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

$$y_j = f(x_j) + \varepsilon_j, \tag{14}$$

where ε_j is assumed to be a sample drawn from $\mathcal{N}(0, \bar{\sigma}^2)$. [4, 8] 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

$$\mathbb{E}_{d_{tr}}\left[\left(f(x) + \varepsilon - f_e(x; d_{tr})\right)^2\right] = \text{bias}^2_{d_{tr}} + \mathbb{V}(f_e) + \bar{\sigma}^2 \tag{15}$$

where

$$\text{bias}_{d_{tr}} = \mathbb{E}_{d_{tr}}\left[f_e\right] - f(x) \tag{16}$$

and

$$\mathbb{V}(f_e) = \mathbb{E}_{d_{tr}}\left[\left(\mathbb{E}_{d_{tr}}\left[f_e\right] - f_e(x; d_{tr})\right)^2\right] \tag{17}$$

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

Introduction - DLDMD

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, we call this method Deep Learning Enhanced DMD (DLDMD). This is implemented by defining an encoder $\mathcal{E} : \mathbb{R}^{N_S} \to \mathbb{R}^{N_O}$ and decoder $\mathcal{D} : \mathbb{R}^{N_O} \to \mathbb{R}^{N_S}$ composed of dense layers such that

$$(\mathcal{D} \circ \mathcal{E})(\boldsymbol{x}) = \boldsymbol{x} \tag{18}$$

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.

Introduction - The loss function

A natural choice considering these constraints is given by

$$\mathcal{L} = \alpha_1 \mathcal{L}_{\text{recon}} + \alpha_2 \mathcal{L}_{\text{dmd}} + \alpha_3 \mathcal{L}_{\text{pred}} + \alpha_4 \left|\left| \boldsymbol{W}_g \right|\right|_2 \tag{19}$$

where

$$\mathcal{L}_{\text{recon}} = \frac{1}{N_T + 1} \sum_{j=1}^{N_T+1} \left|\left| \boldsymbol{y}_j - (\mathcal{D} \circ \mathcal{E})(\boldsymbol{y}_j) \right|\right|_2, \tag{20}$$

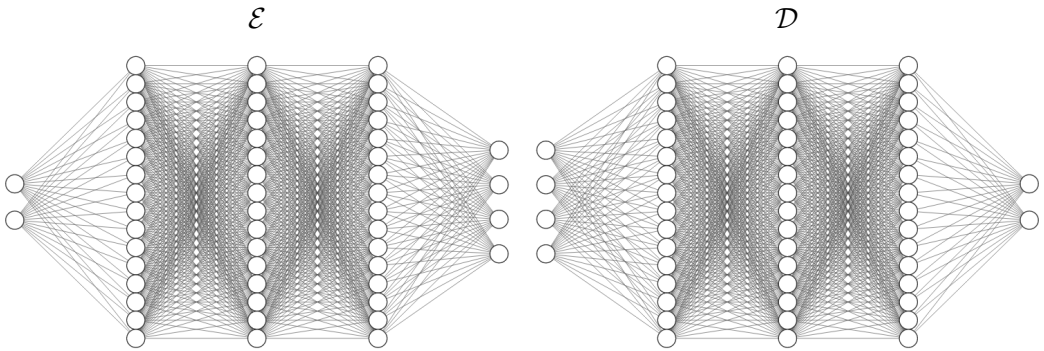
$$\mathcal{L}_{\text{dmd}} = E_r(\boldsymbol{K}_O), \tag{21}$$

$$\mathcal{L}_{\text{pred}} = \frac{1}{N_T} \sum_{j=1}^{N_T} \left|\left| \boldsymbol{y}_{j+1} - \mathcal{D}\left(V T^j V^{-1} \mathcal{E}(\boldsymbol{x})\right) \right|\right|_2, \tag{22}$$

Recall that N_T is the number of measurements in our time-series data after the first.

Introduction - The Network Architecture

Example of DLDMD newtork with $N_S = 2$, $N_O = 4$, and $N_L = 3$ where every hidden layer has 16 neurons. 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.



Introduction - The Dilemma

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

$$L(\boldsymbol{x}; \mathbf{A}, \sigma, \mathbf{b}) = \sigma(\mathbf{A}\boldsymbol{x} + \mathbf{b}) \tag{23}$$

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 $\sigma : \mathbb{R}^{N_O} \to \mathbb{R}^{N_O}$. 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. 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

$$Q_{n+1} = P(Q_n) \tag{24}$$

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.

Introduction - Information Theory

In the interest of examining convergences, one might consider the following limit

$$\lim_{n\to\infty} ||Q_{n+1} - Q_n||_2 \tag{25}$$

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.

Kullback-Leibler Divergence - Entropy

A measure of the uncertainty was proposed in 1948 and this measure is known as *informational entropy* [7] or, simply, entropy. For a continuous distribution with density function $f(x)$, the entropy is given by

$$h[f] = \mathbb{E}\left[-\log(f(x))\right] = -\int_X f(x)\log(f(x))\ dx \tag{26}$$

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

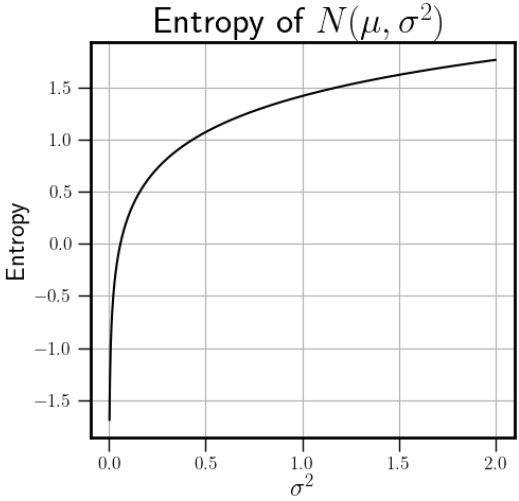
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) \tag{27}$$

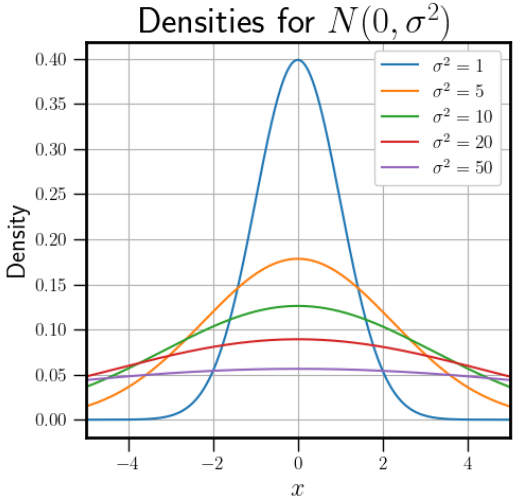
and the entropy integral for this evaluates to

$$h[f] = \frac{1}{2}\left(\log(2\pi\sigma^2) + 1\right) \tag{28}$$

which only depends on the variance.

Kullback-Leibler Divergence - Entropy example





Kullback-Leibler Divergence - 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 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 [6]. 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

$$D_{KL}(P \parallel Q) = \sum_{x \in X} P(x) \log \left(P(x)/Q(x) \right) \tag{29}$$

In our cases, we are be dealing with a continuous random variable with probability density functions p and q, for which the corresponding KLD formula is

$$D_{KL}(P \parallel Q) = \int_X p(x) \log \left(p(x)/q(x) \right) \, dx \tag{30}$$

Kullback-Leibler Divergence - Statistical Distance

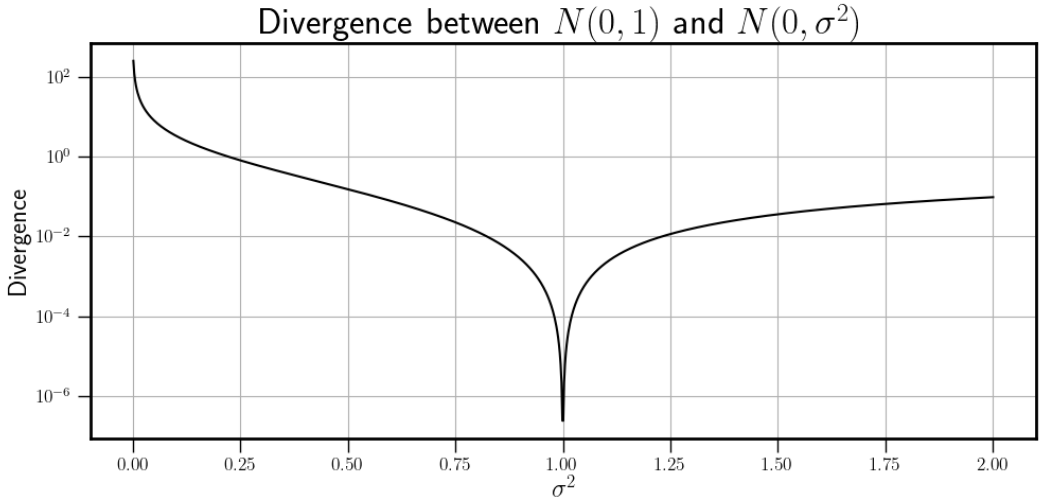
The reason we can consider this a "distance" is that the KLD is non-negative, $D_{KL}(P \parallel Q) \geq 0$, which is known as Gibbs inequality and only equals 0 when $P = Q$ almost everywhere. As such, the distance between P and Q is 0 when they are the "same" distribution and some positive number 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 27; the divergence between the distributions is

$$D_{KL}(p \parallel q) = \frac{1}{2}\left(\ln\left(\frac{\sigma_2^2}{\sigma_1^2}\right) + \frac{\sigma_1^2}{\sigma_2^2} + \frac{(\mu_1 - \mu_2)^2}{\sigma_2^2} - 1\right), \tag{31}$$

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

$$D_{KL}(p \parallel q) = \frac{1}{2}\left(\ln \sigma_2^2 + \frac{1}{\sigma_2^2} - 1\right), \tag{32}$$

Kullback-Leibler Divergence - normal distribution example



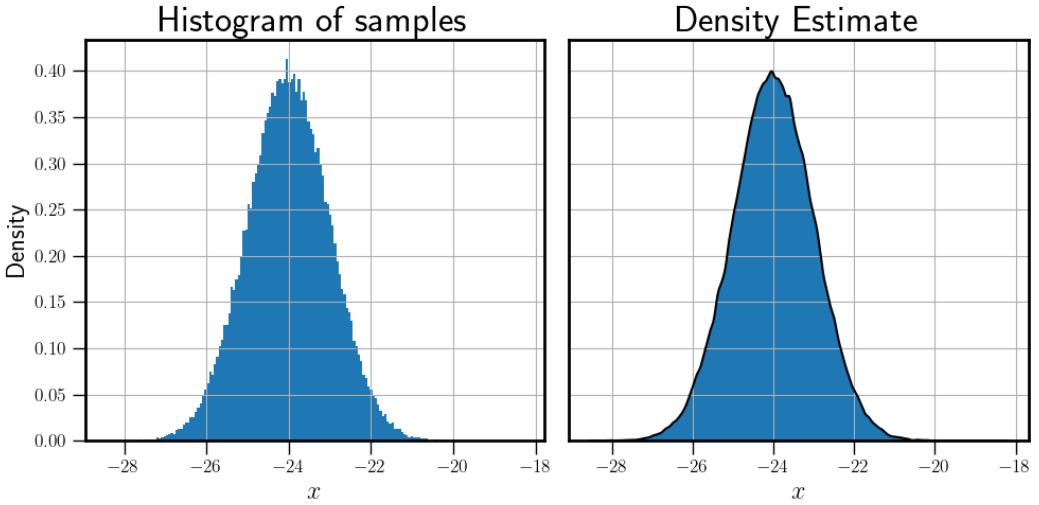
Kullback-Leibler Divergence - Kernel Density Estimation

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

$$\hat{f}(x; K, h) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{33}$$

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 [3]; 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 affliated histogram. We choose h using the Improved Sheather-Jones algorithm [2], which algorithmically finds an optimal bandwidth using the data itself.

Kullback-Leibler Divergence - KDE example



Kullback-Leibler Divergence - Measuring Entropy Flow

Should any given training cause the machine to converge to a fixed state, then for each layer we should have that

$$\mathbf{W}_{l,\mathcal{E}}^{(n)}, \mathbf{W}_{l,\mathcal{D}}^{(n)} \to \mathbf{W}_{l,\mathcal{E}}^{*}, \mathbf{W}_{l,\mathcal{D}}^{*} \text{ as } n \to \infty \tag{34}$$

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

$$\mathbf{W}_{l,\mathcal{E}}^{(n)} = \mathbf{W}_{l,\mathcal{E}}^{*} + \mathbf{W}_{l,\mathcal{E}}^{(n),f}, \ \mathbf{W}_{l,\mathcal{D}}^{(n)} = \mathbf{W}_{l,\mathcal{D}}^{*} + \mathbf{W}_{l,\mathcal{D}}^{(n),f}, \tag{35}$$

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.

Using first-order differencing, we can study the affliated detrended matrices $\delta\mathbf{W}_{l,\mathcal{E}}^{(n)}$ where

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

Kullback-Leibler Divergence - Consecutive Distributions

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 now find the KLD between consecutive distributions to generate the following sequence:

$$D = \left\{ D_{KL}\left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E-2} \tag{36}$$

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

Kullback-Leibler Divergence - Implementation details

After the training of each model for N_E epochs, the data that we have to play with is

$$\mathbf{W}_{\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_L}, \mathbf{W}_{\mathcal{D}} = \left\{ \mathbf{W}_{l,\mathcal{D}}^{(n)} \right\}_{n=1,l=1}^{N_E,N_L}, \tag{37}$$

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

$$\delta\mathbf{W}_{l,\mathcal{E}} = \left\{ \mathbf{W}_{l,\mathcal{E}}^{(n+1),f} - \mathbf{W}_{l,\mathcal{E}}^{(n),f} \right\}_{n=1}^{N_E-1}, \tag{38}$$

these matrices are flattened and their entries are used as the data for KDE.

Kullback-Leibler Divergence - Implementation details cont.

The kernel used is the Epanechnikov kernel, defined as

$$K(x) = \begin{cases} \frac{3(1-x^2)}{4}, & |x| < 1 \\ 0, & |x| \geq 1 \end{cases}, \tag{39}$$

which is optimal in the mean-square error sense. From this, we obtain the set of density estimates for each detrended matrix

$$p_{l,\mathcal{E}} = \text{KDE}(\delta \mathbf{W}_{l,\mathcal{E}}) = \left\{ p_{l,\mathcal{E}}^{(n)}(w) \right\}_{n=1}^{N_E - 1} \tag{40}$$

where w is the weight value itself and $p_{l,\mathcal{E}}(w)$ is the approximate density associated with that weight value. Given these density approximations, we can now compute what we are truly interested in:

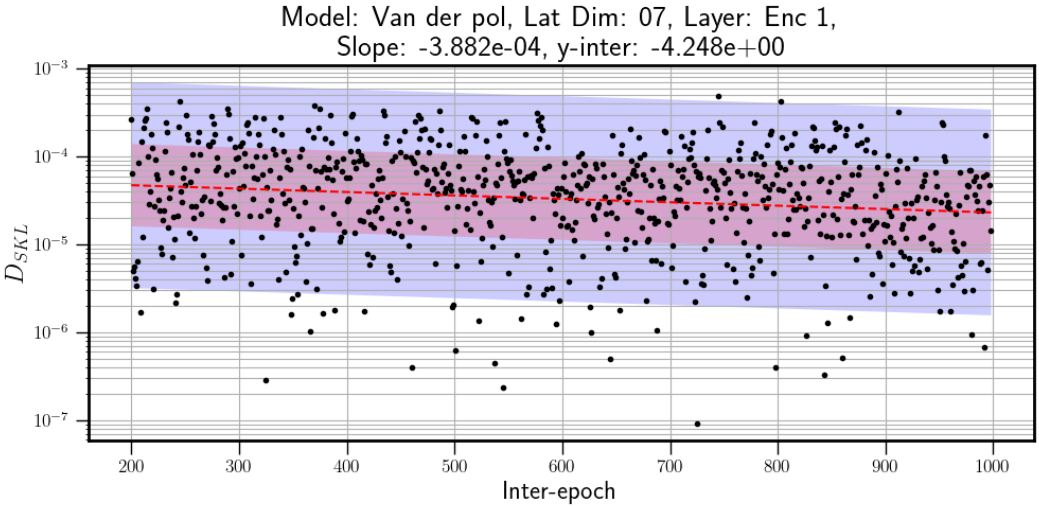
$$D_l = \left\{ D_{SKL} \left(p_{l,\mathcal{E}}^{(n+1)} \middle\| p_{l,\mathcal{E}}^{(n)} \right) \right\}_{n=1}^{N_E - 2} \tag{41}$$

Unlike in the first mention, we instead use the SKLD instead of the KLD.

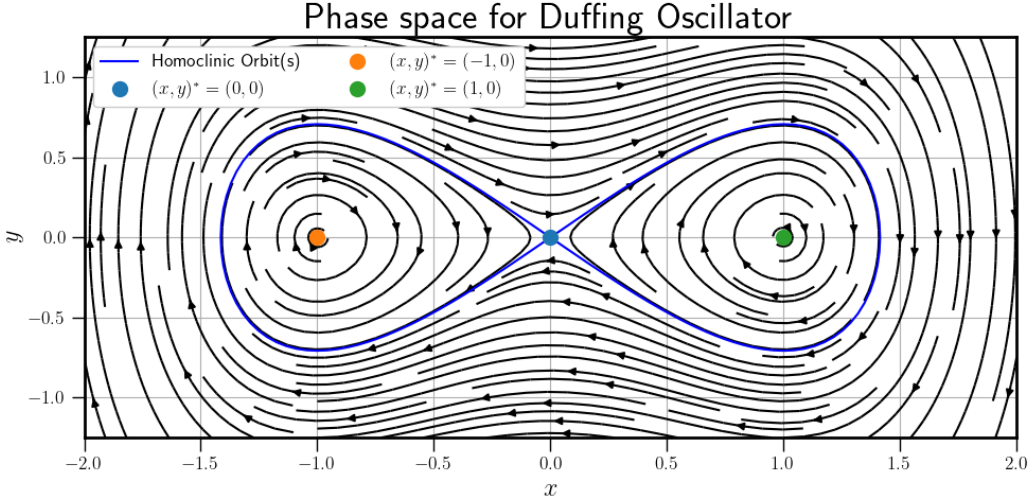
Kullback-Leibler Divergence - Implementation details cont.

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. 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. We 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. 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.

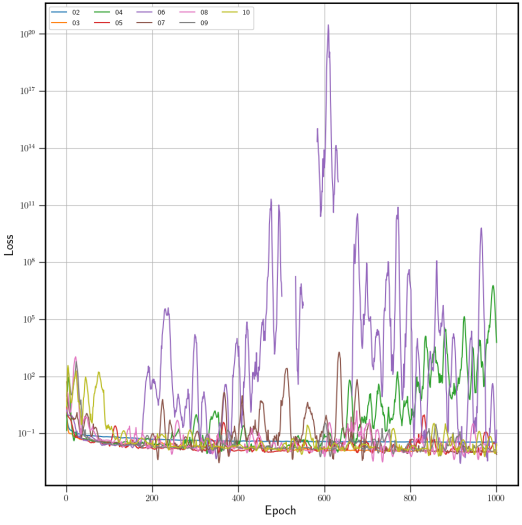
Kullback-Leibler Divergence - Example fitting

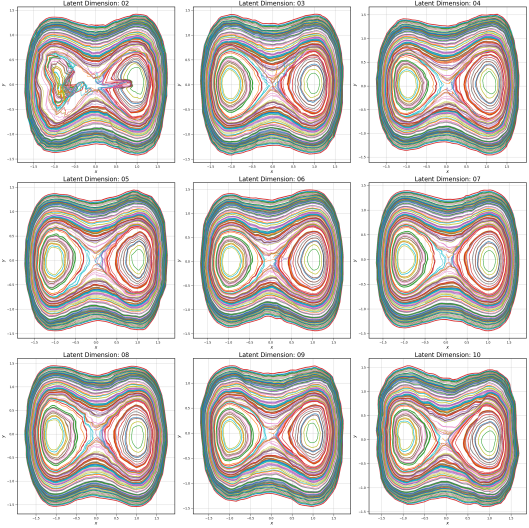


Results - Duffing

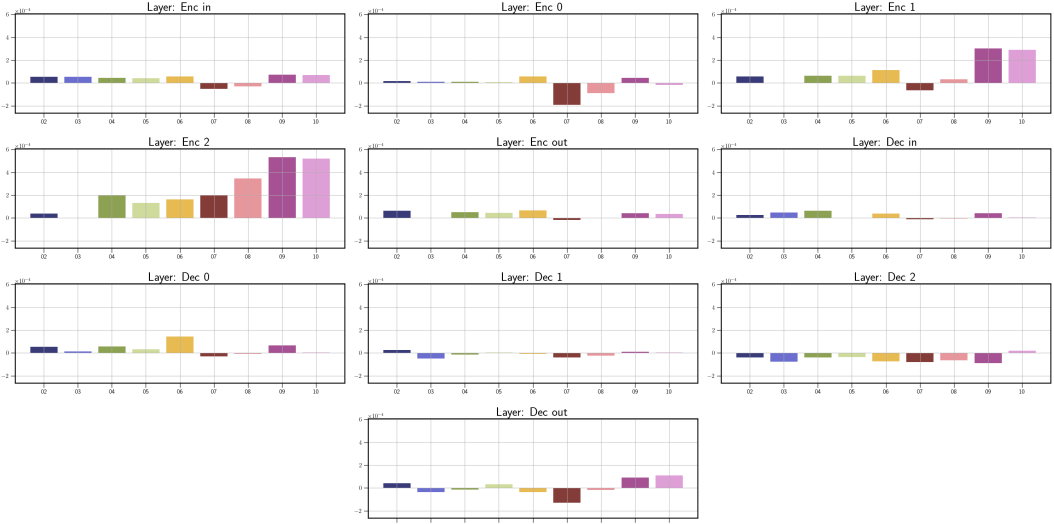


Results - Duffing loss curves and phase space

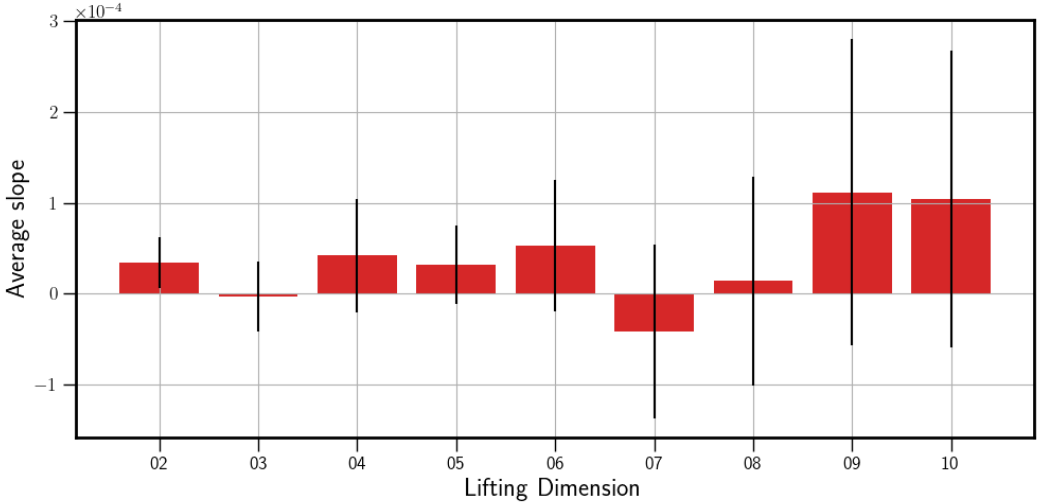




Results - Duffing linear fit slopes

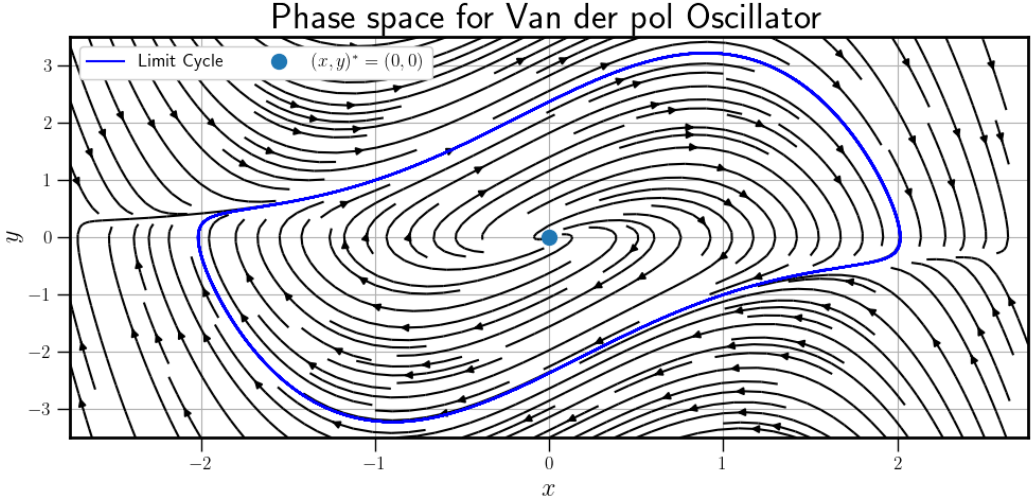


Results - Duffing linear fit slope averages and variances

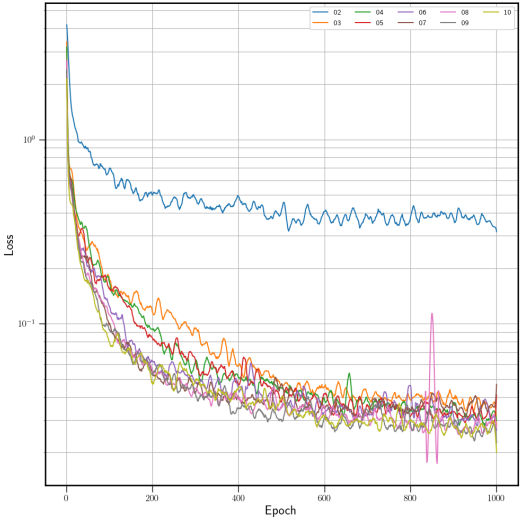


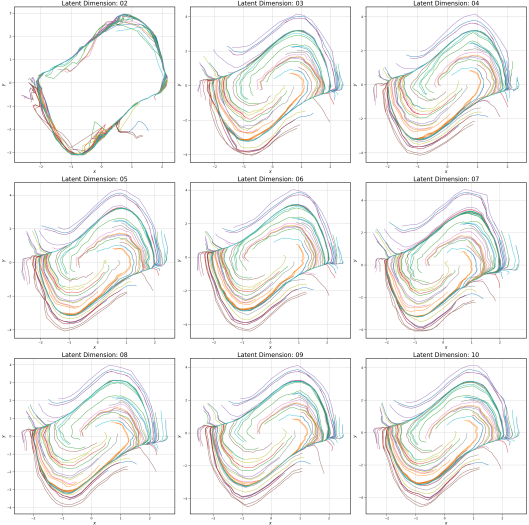
Results - Duffing notes

Results - Van der Pol

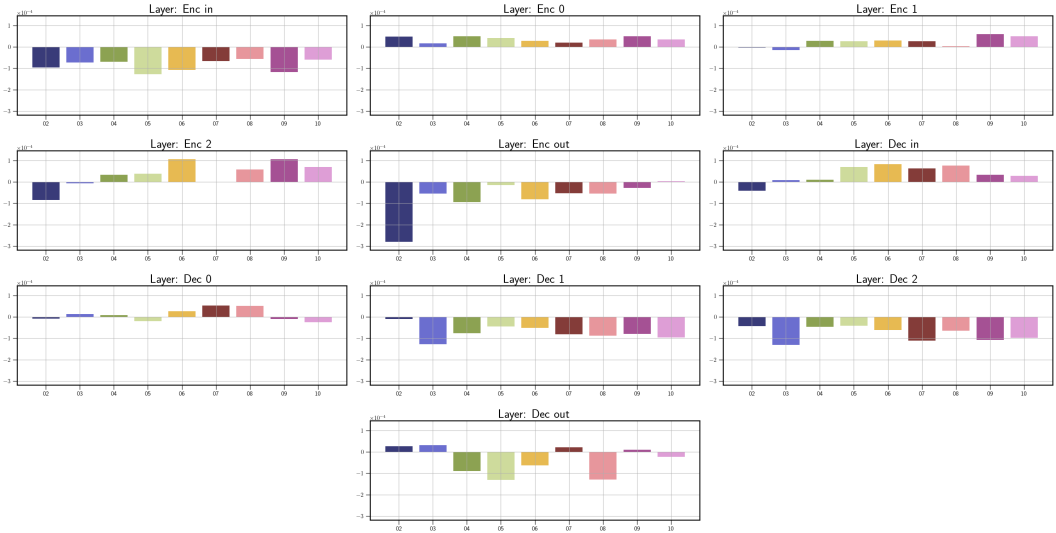


Results - Van der Pol loss curves and phase space

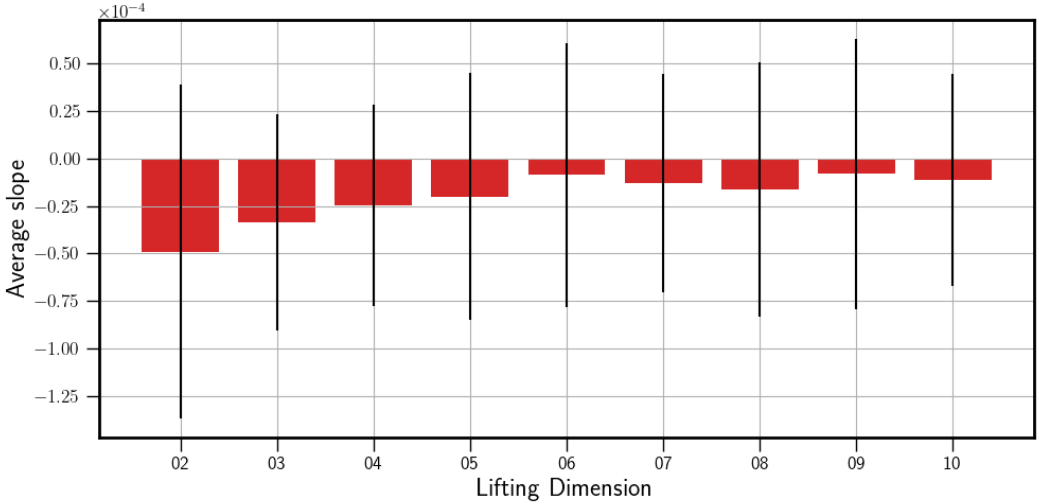




Results - Van der Pol linear fit slopes



Results - Van der Pol linear fit slope averages and variances



Results - Van der Pol notes

Discussion

References

D.J. Alford-Lago, C. W. Curtis, A. T. Ihler, and O. Issan.
Deep Learning Enhanced Dynamic Mode Decomposition.
2022.

Z. I. Botev, J. F. Grotowski, and D. P. Kroese.
Kernel density estimation via diffusion.
The Annals of Statistics, 38(5):2916 – 2957, 2010.

V. A. Epanechnikov.
Non-parametric estimation of a multivariate probability density.
Theory of Probability & Its Applications, 14(1):153–158, 1969.

Aurelien Geron.
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.
O'Reilly Media, Inc., 2nd edition, 2019.

B.O. Koopman.
Hamiltonian systems and transformations in Hilbert space.
Proc. Nat. Acad. Sci., 17:315–318, 1931.

The End!

Thank you for your time!
Questions?