Part III Essays Easter 2020

E763G

Attach to Front of Essay

Essay Number:

Essay Title:

Neural Networks to Linearise Nonlinear Dynamics

May 2020

"To date, we know of only one species that manipulates fluids through knowledge of the Navier-Stokes equations. [...] At the beginning of a new millennium, with increasingly powerful tools in machine learning and data-driven optimization, we are again learning how to learn from experience." – Brunton et al. [2020]

Abstract

Machine learning techniques have brought about great advances in fields ranging from computer vision to finance. In fluid dynamics, a challenge lies in the many degrees of freedom and the inherent nonlinearity of the governing equations. Neural networks may offer a way to solve this problem by finding (nonlinear) transformations to a coordinate space where the time evolution of a nonlinear system appears linear. This approach is closely related to Koopman operator theory. In this essay, I will first review some general concepts in machine learning, Koopman operator theory and their connection. Then I will present my own results from working on the nonlinear pendulum using an autoencoder neural network. Prediction accuracy is high where the input is close to the training data and the network predicts nonlinear behaviour even though the time-evolution layer is linear. However, the network does not extrapolate to other regions of phase space. The results show that in using neural networks for Koopman analysis, extrapolation beyond the training data poses a serious challenge.

Contents

1 Introduction and Background

Machine learning techniques have been applied in many different fields with great success. In this essay, I would like to take a closer look at the possible applications of machine learning to fluid dynamics and dynamical systems. Specifically, the essay focuses on a method combining operator theory and neural networks. The operator theoretical part is based on the work by Bernard Koopman [1931], which focused on the linear operator later named after him and its relation to properties of dynamical systems. Recently, it has been proposed to use neural networks to retrieve the Koopman operator of dynamical systems from time evolution data [Yeung et al., 2019]. This method is applied to non-linear dynamical systems with the goal to allow linear analysis and future state prediction.

The first section, going from general to more specific, will give an overview of different concepts in machine learning. This, together with the operator-theoretical part, will lay the groundwork for an investigation of Koopman analysis. In the second section, Koopman analysis with neural networks will first be described; then my own analytical and numerical results will be presented. These will be discussed in the last section together with possibilities of future investigation and challenges faced by Koopman analysis with neural networks.

1.1 Machine Learning for Fluid Mechanics

Machine learning can be thought of as the problem of finding associations between inputs $\mathbf{x}$, outputs $\mathbf{y}$, the parameters of a learning machine $\mathbf{w}$ and its structure $\mathbf{\Phi}$ (cf. Brunton et al. [2020]). The learning process is stochastic in that the samples ($\mathbf{x}$, $\mathbf{y}$) are drawn from a probability distribution $p(\mathbf{x}, \mathbf{y})$. The process can be thought of as the minimisation of the *risk functional*

$$R(\mathbf{w}) = \int L(\mathbf{y}, \mathbf{\Phi}(\mathbf{x}, \mathbf{y}, \mathbf{w}))p(\mathbf{x}, \mathbf{y})d\mathbf{x}d\mathbf{y} \tag{1}$$

where L is the so-called *loss*, encoding the learning objectives. The learning process is therefore the minimisation (by variation of the learning machine's parameters) of the loss arising from input-output pairs, integrated over the probability of these pairs.

1.1.1 Supervised and Unsupervised Learning

Learning problems can be categorised into supervised and unsupervised processes. The difference is whether additionally to the training input, there is information about the desired output for each input. This describes supervised learning, while in unsupervised learning, the output may be evaluated using some global criteria which are not specific to the input. The discussion in this section is mostly summarised from the review of Brunton et al. [2020].

In **supervised learning**, a simple loss function of inputs $\mathbf{x}$, the outputs $\mathbf{y}$, and the

trainable parameters $\mathbf{w}$ and the structure of the learning machine $\mathbf{\Phi}$ is

$$L(\mathbf{y}, \mathbf{\Phi}(\mathbf{x}, \mathbf{y}, \mathbf{w})) = ||\mathbf{y} - \mathbf{\Phi}(\mathbf{x}, \mathbf{y}, \mathbf{w})||^2 \tag{2}$$

In the training phase, the task is to minimise the risk functional from equation (1) by varying the parameters $\mathbf{w}$, while the scientist can fine-tune the structure $\mathbf{\Phi}$. A more sophisticated loss function can involve additional constraints such as a penalty on large parameters or a different weighting of errors (e.g., a p-norm for some input-output pairs instead of the 2-norm above).

Examples for supervised learning processes include the original *perceptron* proposed by Rosenblatt [1958], which classifies inputs in two categories all the way to modern neural networks capable of recognising street signs [Ciregan et al., 2012]. Neural networks and applications to fluid mechanics will be discussed below.

In **unsupervised learning**, there is no explicit input-output training data. Rather, the neural network is trained to perform some operation on the input data to minimise a loss function encoding an abstract goal. An example for this would be data reduction with k-means clustering, which partitions data into k groups (Voronoi cells in data space). The reference points of each group can then be used to represent the data, effectively coarse-graining the data. This is achieved by varying the parameters $\mathbf{w}$ of the learning machine $\mathbf{\Phi}(\mathbf{x}, \mathbf{w})$ which maps the data $\mathbf{x}$ to the cluster centres. For this, the loss

$$L(\mathbf{\Phi}(\mathbf{x}, \mathbf{w})) = ||\mathbf{x} - \mathbf{\Phi}(\mathbf{x}, \mathbf{w})||^2 \tag{3}$$

is employed. Another method, which is very important for the subsequent sections, is dimensionality reduction using so called autoencoders. An autoencoder can be described as a pair of functions, the encoding $\mathbf{\Phi}$ and the decoding $\mathbf{\Phi}^{-1}$, where for the dimension of $\mathbf{\Phi}(\mathbf{x})$ is lower than that of the data $\mathbf{x}$. The learning problem consists in minimising the loss

$$L(\mathbf{\Phi}, \mathbf{\Phi}^{-1}) = ||\mathbf{\Phi}^{-1}(\mathbf{\Phi}(\mathbf{x})) - \mathbf{x}||^2 \tag{4}$$

This effectively forces the data through an "information bottleneck" and finds an accurate lower-dimensional description of high-dimensional data. For a concrete example see Figure 1, showing a more complex autoencoder architecture.

1.1.2 Neural Networks

Artificial Neural Networks are a common method to implement learning machines, both supervised and unsupervised. They consist of neurons, which, like their natural counterparts, can transmit signals in the form of numbers. Neurons are organised in layers. The neurons of two layers are connected with links that amplify or dampen the signal (in- or decrease the number). This behaviour is encoded in the weights associated with each link.

One way to think about the connection between the neurons in two layers is in terms of matrices. Consider two layers of n neurons each, where every neuron in the second layer can receive the value in any of the neurons in the first layer, multiplied by the weight. These weights can be written in an $n \times n$ matrix. If we write the first layer neuron values in an n-dimensional vector, in a very simple neural network, the second layer could just be this vector multiplied by the weights matrix.

Two more components characterise a neural network: Most importantly, to produce non-linear behaviour (e.g., to make a *decision* about which category some data input belongs to, similar to the street sign example), there needs to be a nonlinear *activation function*. The activation function takes as input the signal from the preceding neuron weighted as described above. The output depends on the function. A simple example is that of the rectified linear unit (ReLU) $f(x) = max(0, x)$. An identity activation function is possible (and was actually described above, where the first layer was just multiplied the weights matrix). Lastly, a constant vector called *bias* can be added to each layer.

While the value of biases and weights in the neural network make up the trainable parameters, the choice of an activation function, the number of neurons in each layer (its *width*) and the number of layers lies with the scientist and depends on the application.

To summarise, the output $G_j(\mathbf{x})$ of a neuron with input vector $\mathbf{x}$, in-going weights $\mathbf{w_j}$, bias $\mathbf{b_j}$ and activation function f can be written as

$$G_j(x) = f(\mathbf{w_j}^T\mathbf{x} + \mathbf{b_j}) \tag{5}$$

The power of neural networks lies in the result, shown by Cybenko [1989], that they can approximate arbitrary continuous functions with compact support. Specifically, in his *universal approximation theorem*, Cybenko showed that finite sums of neuron outputs $G_j(\mathbf{x})$ are dense in the space of continuous function on the unit hypercube $C(I_n)$, i.e., that for all $\epsilon > 0$ and all $g \in C(I_n)$ there is a $G = \sum_{j=0}^{N} G_j(\mathbf{x})$, such that

$$|g(\mathbf{x}) - G(\mathbf{x})| < \epsilon \; for \; all \; \mathbf{x} \; in \; I_n \tag{6}$$

In the neural network architecture, this means that for any desired function, there is a combination of finitely many neurons with their respective weights and biases, such that the function is approximated by the network to arbitrary accuracy. The result was first shown for sigmoidal activation functions, but later extended to any piecewise continuous locally bounded function that is not polynomial (esp. the ReLU function used in this work) by Leshno et al. [1993]. Importantly, the theorem assures existence of such neurons, but not the practical feasibility of learning them.

1.1.3 Applications to Fluid Mechanics

In fluid mechanics, simulations are often expensive as many flow problems span multiple scales and high numbers of dimensions. A potentially useful application of machine learning techniques therefore consists in reducing dimensionality or complexity of the problem. The above mentioned k-means clustering is one such technique. In one example, it has successfully reduced the number of data points necessary to describe the dynamics of a mixing layer Kaiser et al. [2014]. This essay focuses on another method, closely related to that of dimensionality reduction with autoencoders. However, the goal is not only to reduce dimensionality, but also to make the dynamics of the problem more tractable by linearising nonlinear processes.

1.2 Koopman Operator Theory

In a nutshell, the Koopman operator is the linear operator evolving *all* observables of a dynamical system in time. Already for a one-dimensional dynamical system there are infinitely many observables (think polynomials of arbitrary degree in the state space variable), therefore the Koopman operator is infinite-dimensional. Thus, while the state-space picture to describe dynamical systems is often low-dimensional, but nonlinear, the Koopman-description is always high-dimensional but linear. The further details given below follow the work of Mezić [2013] and Arbabi [2018].

1.2.1 The Koopman Operator

Bernard Koopman [1931] first proposed the study of the linear operator which is today named after him. Arbabi [2018] relays how Koopman first wrote about its properties in the context of dynamical systems and ergodic theory. Consider a dynamical system consisting of the state space $S \subset \mathbb{R}^n$, and a discrete time-evolution given by the map $T : S \longrightarrow S$ which satisfies

$$x_{t+1} = T(x_t) \tag{7}$$

for all $x \in S$ and all $t \in \mathbb{Z}$. The set of observable functions $g : S \longrightarrow \mathbb{R}$ on the state space form a vector space on which we can define the Koopman operator $U : S \longrightarrow S$ such that

$$Ug(x_t) = g \circ T(x_t) = g(x_{t+1}) \tag{8}$$

This means, that the Koopman operator evolves the observables in time. The Koopman operator is linear, because for two observables g_1 and g_2

$$U(g_1 + g_2)(x) = (g_1 + g_2) \circ T(x) = g_1 \circ T(x) + g_2 \circ T(x) = Ug_1(x) + Ug_2(x) \tag{9}$$

The vector space of all observable functions is infinite-dimensional. It may therefore be helpful to picture the Koopman operator as an infinite-dimensional matrix, which maps the observables from time step t to time step $t + 1$.

1.2.2 Linear Expansion of Observables

As for many linear operators, it is interesting to study its eigenfunctions. These are the functions $f : S \longrightarrow \mathbb{R}$ satisfying

$$Uf(x) = e^{\lambda \Delta t} f(x) = e^{\lambda} f(x) \tag{10}$$

for some $\lambda \in \mathbb{C}$ and in a dynamical system with time step $\Delta t = 1$. If these eigenfunctions span the space of all observables, any observable can be expanded in eigenfunctions f_k such that

$$Ug(x) = \sum_{k=0}^{\infty} g_k e^{\lambda_k} f_k(x) \tag{11}$$

where g_k are the coefficients of the eigenfunctions. Koopman also observed, that for a Hamiltonian system, the Koopman operator is unitary and hence its eigenvalues e^{λ} are of absolute value unity.

1.2.3 Dynamic Mode Decomposition

Dynamic Mode Decomposition (DMD) is a numerical technique related to Koopman analysis. DMD is motivated by the fact that even though fluid systems are governed by the Navier-Stokes equations, i.e., infinite-dimensional partial differential equations, the main properties of some problems can be represented in a finite-dimensional way. For example, in the case of the flow past a cylinder in two dimensions, three ordinary differential equations describe fluid behaviour (cf. Noack et al. [2003]).

DMD, as introduced by Schmid [2010], aims at identifying spatial structures in observable data (most commonly a measured or simulated flow field) which evolve linearly. These structures, together with the linear operator evolving them in time, are identified via the following algorithm, adapted from Tu et al. [2013]:

1. Let $x_1, x_2, ..., x_N$ be a temporal sequence of data vectors of dimension n. Arrange them into $n \times (N-1)$ matrices

$$X = [x_1, x_2, ..., x_{N-1}], \qquad Y = [x_2, x_3, ..., x_N] \tag{12}$$

2. Compute a Singular Value Decomposition (SVD) of X, given by

$$X = U \Sigma V^* \tag{13}$$

with U an $n \times r$ unitary, Σ an $r \times r$ diagonal, V^* an $r \times (N-1)$ unitary matrix and r the rank of X

3. Define the matrix

$$A = U^* Y V \Sigma^{-1} \tag{14}$$

4. Find eigenvalues and eigenvectors of A

$$Av = \lambda v \tag{15}$$

5. The DMD mode corresponding to the eigenvalue λ is then given by

$$\phi = Uv \tag{16}$$

Note that the definition in 14 can be used to show

$$UAU^*X = Y \tag{17}$$

We can interpret UAU^* as the operator evolving the state vectors in time, which also explains the form of the DMD modes in the fifth step:

$$UAU^*\phi = UAU^*Uv = \lambda Uv = \lambda\phi \tag{18}$$

The DMD modes are just the eigenvectors of the linear operator UAU^*. This result brings us already quite close to finding a Koopman operator: What we find using Dynamic Mode Decomposition is a linear operator evolving the system of state space variables in time. The main caveat is that so far, we have only considered the first N snapshots of the system's evolution. We cannot be sure these are enough to predict the system's behaviour at later times.

Also, DMD is a linear algorithm: The matrix UAU^* can therefore be interpreted as the restriction of the Koopman operator to those observables that are the state space variables themselves. Rowley et al. [2009] showed that, under certain conditions, this is true and DMD modes are a subset of Koopman modes. This makes sense as the Koopman operator is infinite-dimensional and every finite set of modes can only be a subset of the Koopman modes. Furthermore, the set of observables must include the set of state variables themselves. The condition for DMD to approximate the Koopman operator restricted to the state variables is that the input vectors span an invariant subspace under Koopman operator application [Li et al., 2017]. This means that the Koopman operator should map elements of the subspace into the subspace itself.

1.2.4 Extended DMD

DMD was extended by allowing for observables other than the system state itself, a technique first proposed by Williams et al. [2015]. In extended DMD (EDMD), a dictionary of functions $D = \{\Psi_j : S \longrightarrow \mathbb{C} \mid j = 1, ..., p\}$ mapping the state space S to the complex numbers is chosen based on knowledge of physical system properties – e.g., in a fluid context, a velocity square observable $\mathbf{u} \cdot \mathbf{u}$ like in the inertial part of the Navier Stokes equation would be an obvious choice. This dictionary can be thought of as a single, vector-

valued function $\mathbf{\Psi} : S \longrightarrow \mathbb{C}^p$ mapping the state space to a p-dimensional observable space.

Then, with the same snapshot data $x_1, x_2, ..., x_N$ as before, we need to minimise

$$J = \sum_{j=2}^{N} |\mathbf{\Psi}(\mathbf{x_j}) - K\mathbf{\Psi}(\mathbf{x_{j-1}})| \tag{19}$$

as a function of the elements of the matrix K. Note that $\mathbf{\Psi}$ is a vector of functions we chose. When making this choice, it makes sense to use a set of functions which allows us to reconstruct the system's state from the value of $\mathbf{\Psi}$. Then, EDMD allows to lift the system to an observable space, find a linear operator which evolves observables and then go back to state space.

The observable functions themselves can be used to construct an approximation to the Koopman eigenfunctions discussed in section 1.2.2. To do this, the matrix K needs to be diagonalised and its eigenvector matrix multiplied with the vector-valued $\mathbf{\Psi}$. Indeed, Korda and Mezić [2018] established the convergence of the linear operator K found using EDMD to a truncated Koopman operator in the limit of large amounts of data and of large dictionary dimension p. Here, the assumption of an invariant subspace spanned by the observables can be relaxed, albeit at the expense of increasing the dimensionality or number of observables p.

2 Finding the Koopman Operator Using Neural Networks

2.1 Autoencoder Architectures for Koopman Analysis

A challenge faced by EDMD is striking a balance between representational capacity and the risk of overfitting when choosing the above-mentioned dictionary of functions Ψ. When the dictionary is too small, it will not contain enough functions for the resulting Koopman operator to be accurate enough. When the dictionary is too large, there is a risk of overfitting, i.e., selecting functions which only work well on the set of data considered, but perform poorly with another data set from the same dynamical system. This, in turn, would mean that they have failed to actually represent the underlying dynamical system. Neural networks can help to strike this balance by *learning* a suitable dictionary.

Some of the first applications of neural networks to finding Koopman representations of dynamical systems were reported in 2017 by Li et al. [2017], Yeung et al. [2019], Takeishi et al. [2017] and Otto and Rowley [2019]. These authors use a neural network to learn the dictionary functions from the data they analyse. Otto and Rowley [2019] draw an analogy between shallow neural networks and dictionary encoding like in EDMD: While shallow neural networks need exponentially more parameters for the representation of certain kinds of behaviour, dictionaries need more and more functions. Deep neural networks, on the other hand, are much more efficient universal encoders and the authors therefore expect

them to perform better than "shallow", large dictionaries. The authors demonstrate this technique using the examples of the Duffing equation and the cylinder wake, among others.

Similar model systems are also studied by Lusch et al. [2018]. They propose an autoencoder network, which is meant to encode state space variables (lift them to observable space), evolve them with a linear Koopman layer, and then decode them (transform back into state space). This way, an autoencoder is not only used for dimensionality reduction (cf. section 1.1), but also for time evolution of the system's state. Additionally to this autoencoder, they use an auxiliary network, which parametrises the Koopman layer depending on the input. Instead of a large-dimensional, but truncated Koopman operator, they rely on one with only two dimensions. This also means, that the observable dictionary only gives a two-dimensional encoding.

The rationale behind this approach is that in many systems, eigenvalue spectra of the Koopman operator have a continuous part (like in the nonlinear pendulum, as shown below). With the parametrisation of the Koopman operator depending on the input, a continuous-spectrum operator can, in principle, be represented by the network without the need for a large number of dimensions (see section 2.2 for the case of the pendulum). An important open question remains to what extent an autoencoder like the one presented in Lusch et al. [2018] can actually predict the time evolution in parts of state space they were not trained on.

2.2 An In-depth Example: the Nonlinear Pendulum

To illustrate the above, let us consider the case of the pendulum as an example for a nonlinear dynamical system. For small perturbations (i.e., small angles) the sinusoidal dependence of angular acceleration on the angle is often linearised. This approximation is not valid for larger angles and so even the undamped, unforced pendulum is a nonlinear system. The governing equations are given by

$$\frac{dx_1}{dt} = x_2 \tag{20}$$

$$\frac{dx_2}{dt} = -\sin x_1 \tag{21}$$

Physically, x_1 describes the angle the pendulum encloses with the vertical and is defined to vanish at the equilibrium position. Its derivative x_2 is the angular velocity. In this section, I will first derive the Koopman eigenfunctions analytically, then I will present some of my own numerical results obtained with an autoencoder similar to the one used by Lusch et al. [2018] and finally compare the analytic and the numerical approach.

2.2.1 Analytic Derivation of Koopman Eigenfunctions

In this system, energy is conserved. It is given by

$$H = \frac{1}{2}x_2^2 - \cos x_1 \tag{22}$$

A physical pendulum can be in two states: One, where it swings back and forth and never reaches the position at the top corresponding to $x_1 = \pi$. Or another, where it reaches that point and swings through it, changing its phase x_1 from π to $-\pi$ and never changing sign of the velocity x_2. The system does not include dissipation, which means that this behaviour repeats itself indefinitely.

In the first, librating case (i.e., swinging back and forth), we must have $H < 1$, giving closed orbits in phase space. Physically, this corresponds to a situation where, at the motions turning point, where $x_2 = 0$, we must also have $-\pi < x_1 < \pi$. In the second, "swinging-through" case, when $H > 1$, the pendulum does not reach $x_2 = 0$, but remains with $|x_2| > 0$ even at the highest point where $x_1 = \pm\pi$.

Now, if we restrict ourselves to the case when $H < 0$, we can define coordinates analogous to polar coordinates, except that the role of radius will be taken by energy and that of the azimuthal angle by the time as a fraction of the period. More concretely: Conservation of energy allows to calculate the time the pendulum takes to move between two given angles $x_{1,1}$ and $x_{1,2}$ by first expressing angular velocity in terms of energy and position

$$x_2 = \sqrt{2(H + \cos x_1)} \tag{23}$$

and then integrating over the inverse of the angular velocity from start to end of the trajectory.

$$t_{1,2} = \int_{x_{1,1}}^{x_{2,2}} \frac{1}{\frac{dx_1}{dt}} dx_1 = \int_{x_{1,1}}^{x_{2,2}} \frac{1}{\sqrt{2(H + \cos x_1)}} dx_1 \tag{24}$$

Plugging in the angles where angular velocity reaches zero

$$x_{1,1/2} = \pm arccos(H) \tag{25}$$

yields the period T. This integral can be expressed in terms of a complete elliptic integral of the first kind.

We can imagine the $H < 0$ phase space as a set of trajectories with a given energy. Every such trajectory is a closed oval orbit of period T. The position on this orbit can now be specified by the time it takes for a pendulum to get there from a starting point we

may choose. We can therefore define new coordinates for the system

$$(x_1, x_2) \longrightarrow (H, \phi) \tag{26}$$

where

$$\phi = 2\pi \frac{t_{1,2}}{T} \tag{27}$$

with $t_{1,2}$ being calculated by inserting integral boundaries 0, x_1 into equation 24. These can be compared to action-angle coordinates, which are useful in various non-dissipative systems where the action is conserved. In these coordinates, the time-evolution of observables is much simpler than in terms of x_1 and x_2: Let $g(x_1, x_2)$ be an observable on the state space, then the transformed function $g(H, \phi)$ behaves under time evolution like:

$$U_t g(H, \phi) = g(H, \phi + 2\pi \frac{t}{T}) \tag{28}$$

where the time step is t. Now, Fourier-transforming the time-dependence, we can expand $g(H, \phi)$ into modes

$$g(H, \phi) = \sum_{j=0}^{\infty} g_j(H) e^{ij\phi} \tag{29}$$

These Fourier modes $f_j(\phi) = e^{ij\phi} = e^{ij2\pi \frac{t}{T}}$ are just the Koopman eigenfunctions we were looking for, because they obey the relation

$$U_t f_j(\phi) = f_j(\phi + 2\pi \frac{t}{T}) = e^{ij(\phi + 2\pi \frac{t}{T})} = e^{ij\phi} e^{ij2\pi \frac{t}{T}} = \lambda_j f_j(\phi) \tag{30}$$

with $\lambda_j = e^{ij2\pi \frac{t}{T}}$. Now, at first sight, it looks like we have found an infinite, but discrete spectrum of Koopman eigenfunctions, parametrised by the positive integers. But this discrete property is only true for a given energy: The eigenvalues depend on the period T, which itself is a function of energy by equation 24. Therefore, we have a continuous spectrum of eigenfunctions. These can be used to expand given observables. In the case of the pendulum, the observables are going to be the state space variables x_1 and x_2 themselves. In the case of x_1, the Koopman modes are just the Fourier coefficients, which take the form

$$\hat{x}_{1,j} = \frac{1}{2\pi} \int_{-\pi}^{\pi} x_1(\phi) e^{ij\phi} d\phi \tag{31}$$

A similar derivation can be found in the work of Govindarajan et al. [2016], where the authors study the properties of a forced system.

With the eigenvalues found above, several predictions about the Koopman operator we hope to find for the nonlinear pendulum can be made. Firstly, all eigenvalues lie on the unit circle in the complex plane. This is consistent with the unitary property of the Koopman operator. Secondly, the eigenvalues will form a spectrum, which is discrete for a given energy but varies continuously with changing energy – we can imagine a kind of Dirac-comb wrapped around the unit circle, where the spacing decreases with increasing period

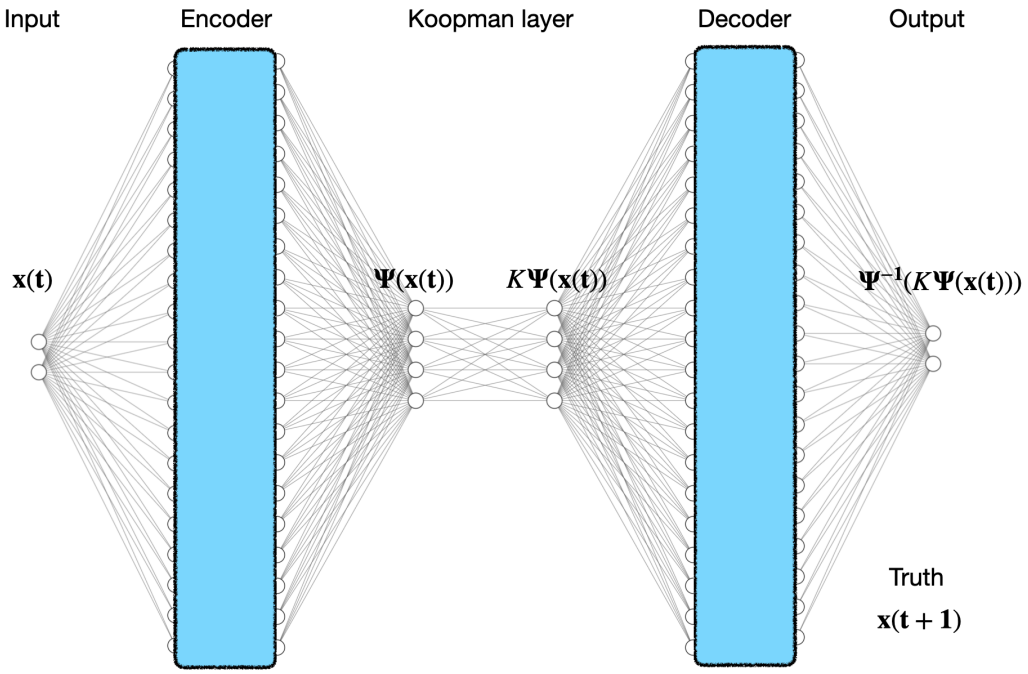


Figure 1: A schematic overview of the neural network architecture employed in this section. The two-dimensional input $\mathbf{x}(t)$ is mapped to an encoded vector $\mathbf{\Psi}(\mathbf{x}(t))$ with arbitrary dimension (four in the Figure). This vector is multiplied by the matrix K and then decoded to give $\mathbf{\Psi^{-1}}(K\mathbf{\Psi}(\mathbf{x}(t)))$. The neural network is trained with the objective to minimise the difference of this prediction and the actual trajectory $\mathbf{x}(t+1)$ (and subsequent timesteps $t+j$).

(increasing energy). Specifically, given a discrete-time data set of the nonlinear pendulum system with time step Δt, the eigenvalues for a given energy (and thereby period T) take the form $e^{2\pi n \frac{\Delta t}{T}}$.

This means that in the case where the Koopman operator is truncated (and we expect it to be truncated in a way that still allows it to predict the future development with reasonable accuracy), the eigenvalues found will correspond to varying integers n and varying periods T. It can further be expected that the eigenvalues depend heavily on the energies of the trajectories in the data.

2.2.2 Constructing a Neural Network

In the following, I will present the numerical work I undertook to study the application of autoencoders to the nonlinear pendulum. The network architecture draws heavily on the work of Lusch et al. [2018], even though it is somewhat simplified.

Setup. Fig. 1 shows the architecture of an autoencoder which first encodes the input (the state space variables, $\mathbf{x} = (x_1, x_2)$), evolves the encoded input in time and then decodes it again. The encoder and decoder can, in principle, have an arbitrary number of

layers and arbitrary widths, symbolised by the light blue boxes. In this work, however, the both consist only of the two densely connected layers shown. They are all of width 80. Similarly, the dimensionality of the encoding output, i.e., the dictionary $\mathbf{\Psi}$ is a design choice. In Fig. 1, this is depicted as four-dimensional. Lusch et al. [2018] use a two-dimensional encoding. In a first step, the encoding dimension is kept the same as the width of encoder and decoder layers (80).

Another choice concerns activation functions used in each layer. While a nonlinear activation function is needed in encoder and decoder layers to allow them to approximate nonlinear functions, the Koopman layer needs to use a linear activation. This ensures its linearity. The nonlinear activation function chosen for encoder and decoder layers is the rectified linear (ReLU) function $f(x) = max(x, 0)$. Similarly, these layers are constructed to have a trainable bias which is added to their output independent of the input. The Koopman layer, in a first step, has no bias added to it.

I implemented the architecture using the Keras extension to the TensorFlow library for Python (cf. Chollet [2015]). The full, commented code, including the architecture and the pendulum ODE solver can be found on Github [essay author, 2020].

Training. The crucial part of the training process is the loss function which the neural network tries to minimise. A simple loss function $\mathscr{L}_{pred}$ which trains the neural network to predict the whole trajectory of n time steps from the initial condition $\mathbf{x}(t_0)$ is given by

$$\mathscr{L}_{pred} = \frac{1}{n} \sum_{j=0}^{n} ||\mathbf{\Psi^{-1}}(K^j \mathbf{\Psi}(\mathbf{x}(t_0))) - \mathbf{x}(t_0 + j)||^2 \tag{32}$$

This function encodes the initial condition, applies the Koopman layer j times, corresponding to j time steps, decodes the result and compares it with the value of the state space variables at time $t + j$. For the network to be trained using this function, it needs to be supplied with the actual trajectory $\mathbf{x}(t_0 + j)$ with $0 \leq j \leq n$ (the *truth* in machine learning terms), which is obtained by numerically solving the ODEs 20 and 21 governing the pendulum's motion.

Additionally, the architecture enforces a sparsity constraint by imposing a penalty on large layer weights via

$$\mathscr{P} = \sqrt{\sum_{i,j} W_{ij}^2} \tag{33}$$

for each layer where the matrix W contains the weights. This is to avoid overfitting the data. This penalty is weighted by a coefficient $\alpha = 10^{-13}$ and added to the loss $\mathscr{L}_{pred}$.

During the training process, on the order of 10^6 trajectories are used. This data is split into three equal parts: training data, validation data and testing data. The training data

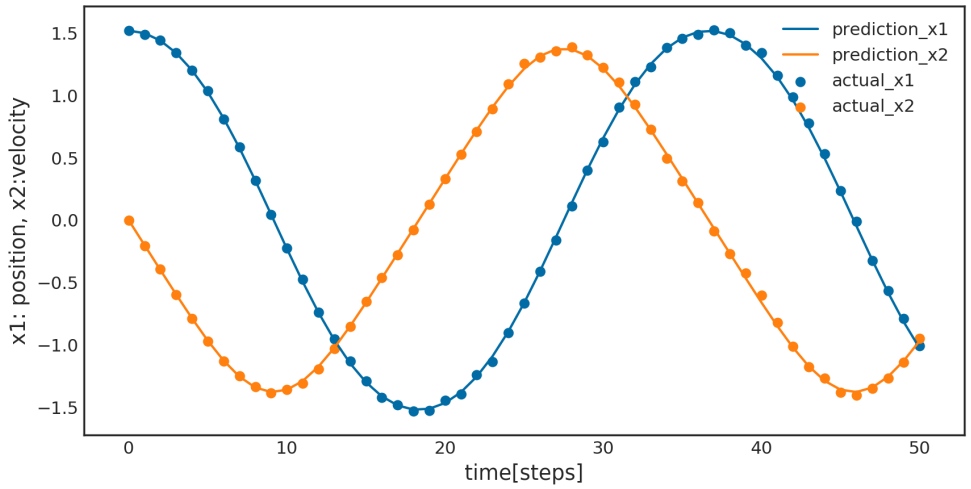


Figure 2: An example of the prediction capacity of a network. It shows the predicted position (blue) and velocity (orange) marked by dots. Solid lines show the actual values obtained from the initial condition by numerical solution of the governing equation. This good prediction accuracy is reproduced over all initial states close to the ones the network was trained with. The dictionary $\mathbf{\Psi}$ is of dimensionality 80.

is given to the network, which minimises the combined loss $\mathscr{L}_{pred} + \alpha \mathscr{P}$ by varying weights and biases. With the network parameters varying, the training data is used several times over, i.e., for multiple *epochs*, typically on the order of 10 to 100.

The validation data is used to monitor how well the model performs on data it has not been trained on (even though in this work, the data in the validation batch is always from the same region of phase space as the one in the training batch). Looking at the validation loss, *hyperparameters* like α can be modified by the scientist to increase the network performance. Other design choices like activation function or the use of biases can also be modified. Lastly, the test batch is used to evaluate this performance on data that was neither used for training before nor for fine-tuning the architecture. Results from this test data is reported in the next section.

2.2.3 Training results

The network in its initial configuration is considerably simpler than the one presented in Lusch et al. [2018], especially since there is no auxiliary network parametrising the Koopman layer according to input (cf. section 2.1). Furthermore, the loss function here contains only two of three terms present in their work. The missing term will be added later. In a first step, the network is trained for 30 epochs. Results from initial conditions within the training data subspace are compared with those outside the training subspace. The latter are found to have very low accuracy. Furthermore, the Koopman operator found has eigenvalues of absolute value lower than unity, contrary to the derivation in section

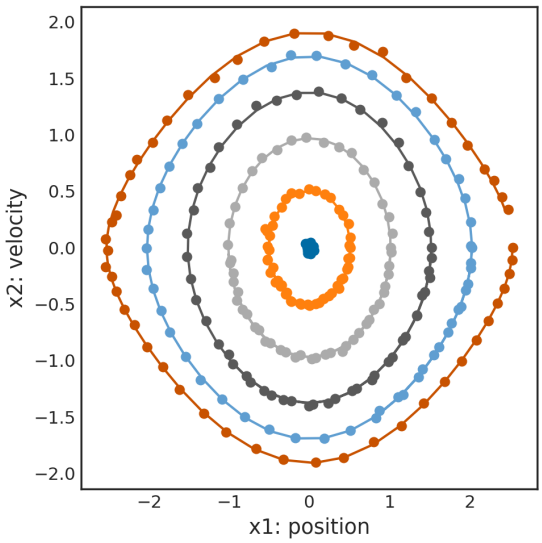


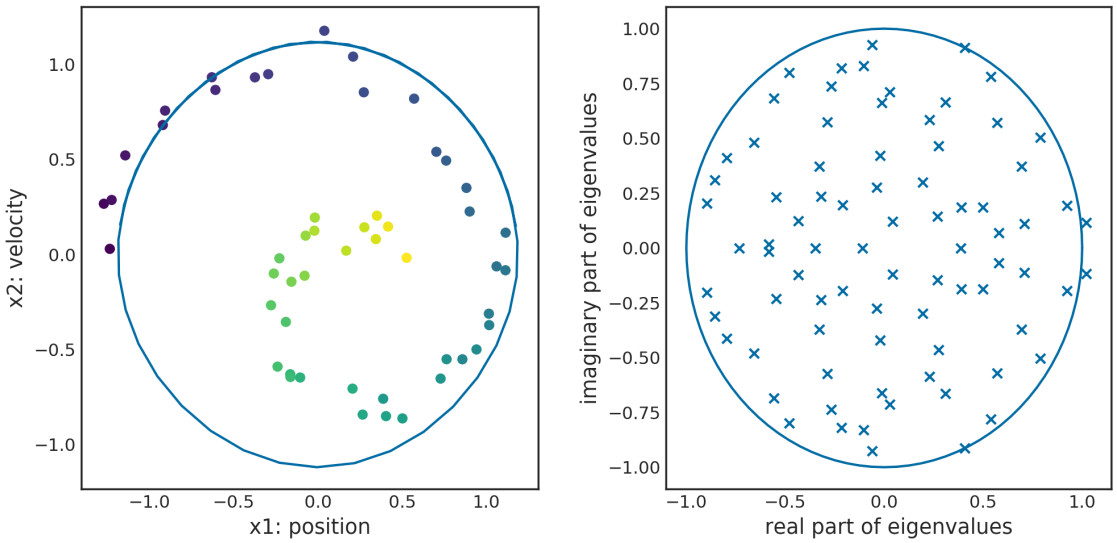
Figure 3: Predictions (dots) and actual values (solid lines) for six example trajectories in phase space. The trajectories start on the positive x-axis and then run clockwise. Different periods of the system depending on initial conditions are visible: All pendulums swing for the same time, but while some perform more than a whole oscillation, the outermost trajectory does not.

2.2.1. Both the additional loss term and the parametrisation of the Koopman operator similar to Lusch et al. [2018] are found not to benefit the prediction of the neural network.

Within training subspace. Fig. 2 shows a first prediction of the network. The network was trained for 30 epochs with data of trajectories starting on the x-axis, i.e., with the pendulum released at some position with zero velocity. As an order of magnitude, the 30 training epochs with 5 10^5 trajectories correspond to around an hour of computing time on a 2,2 GHz Quad-Core Intel Core i7 processor. A prediction for such a trajectory from the test batch is shown to be very accurate in the figure. The same is true over the whole range of possible initial values, as can be seen in Figure 3.

This figure also shows that for large initial elongation, i.e., high energies, the period T increases (cf. equation 24). This is a nonlinear effect, as in the linearised case, the pendulum has a constant period of $T = 2\pi\sqrt{l/g}$ (or in our case just $T = 2\pi$ as we set l and g to be one). The neural network is capable of approximating this nonlinear effect, even though the time evolution layer K is just a linear transformation. This is a promising result for the goal of finding a coordinate transformation to linearise system dynamics.

Beyond training subspace. Even though results for initial conditions on the x-axis are promising, slight deviations from this training set subspace result in large prediction errors. This is exemplified by Figure 4 **a)**, where the trajectory starts slightly off the x-axis (i.e., with a small initial velocity). The prediction quickly starts to deviate from the actual trajectory.



(a) Trajectory for initial condition marginally off training data.

(b) Eigenvalues of the Koopman operator.

Figure 4: Two results that raise doubts about the reliability of network predictions like in Fig. 3. In **a)**, the initial condition is only slightly off the x-axis. Nevertheless, the prediction (dots, with colour encoding time from blue to yellow) deviates more and more from the actual trajectory (solid blue line). Even though the prediction follows a similar pattern as the actual trajectory by moving clockwise, the accuracy is much lower here. In **b)**, eigenvalues of the Koopman operator layer are plotted in the complex plane together with the unit circle. According to the derivation in 2.2.1 the eigenvalues are expected to lie on the unit circle. However, most eigenvalues are actually found to have smaller absolute value.

Figure 5 extends this observation to the whole of the $H < 1$, i.e., librating pendulum subspace. The Figure shows the cumulative error of the prediction over a whole trajectory as the colour of the point where the trajectory originates. It shows that prediction accuracy is good for initial conditions on the training set, but quickly drops off when the trajectory starts elsewhere. Quantitatively, the average error of a predicted trajectory in two variables summed over 30 time steps is around 90. This is only slightly better than a generic prediction, that the trajectory, no matter where it starts, immediately jumps to the origin. This would yield an average error of around 150. However, within the training set, the average error summed over one trajectory is on the order of one.

In a second step, the network is trained with data from all over the right half of the $H < 1$ oval. Training is for 150 epochs, this time with about double the number of trajectories in the training data. This reduces the average prediction error in the whole oval to around 80, but it seems probable that with a sufficiently large training data set which reaches the same density of initial conditions as the training data on the x-axis, the prediction can get much more accurate.

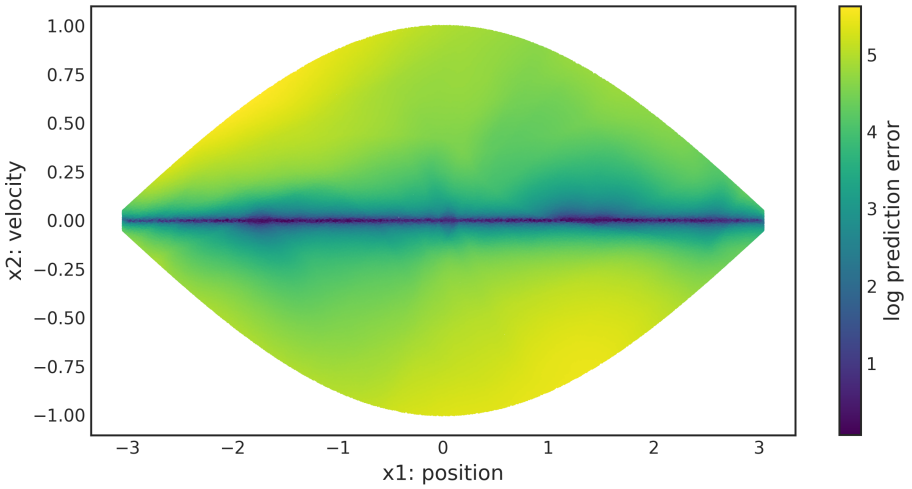


Figure 5: A systematic investigation of the prediction error, this plot shows the logarithm of the prediction error over whole trajectories. Each point corresponds to the initial condition indicated by it's position. The absolute value of the prediction error is summed over the whole trajectory and its log is given by the point's colour. This shows that the accuracy of the prediction quickly drops away from the training data region.

Swing-trough pendulum. The prediction accuracy of this network, trained on the $x_1 > 0$-half-oval, is also tested outside of the $H < 1$ subspace. Figure 6 shows the prediction errors in the case where trajectories start all over the region $[-\pi, \pi]^2$. Importantly, the network does not extrapolate effectively to the left half-oval and performs even worse in the higher-energy regime.

Properties of the Koopman operator. In Figure 4 **b)**, eigenvalues of a matrix-representation of the time evolution layer weights are shown. This layer corresponds to the Koopman layer, so the eigenvalues should be expected to be those of the Koopman operator, derived in section 2.2.1. One of the most basic of their properties is that we expect the eigenvalues to be of absolute value unity. However, the observed eigenvalues almost all lie within the unit circle. This may partly be explained by the fact that around half of the entries of the encoded variable are empty for all inputs (as shown in Figure 7). In this case, many matrix entries would have no significance.

Additional loss term. The term missing in the loss function used up to here compared with the work of Lusch et al. [2018] takes the following form:

$$\mathscr{L}_{lin} = \frac{1}{n} \sum_{j=0}^{n} ||(K^j \mathbf{\Psi}(\mathbf{x}(t_0)) - \mathbf{\Psi}(\mathbf{x}(t_0 + j))||^2 \tag{34}$$

This is very similar to the loss term $\mathscr{L}_{pred}$ in equation 32, except that the difference is now taken of the non-decoded Koopman layer output and the encoded trajectory. In $\mathscr{L}_{pred}$,

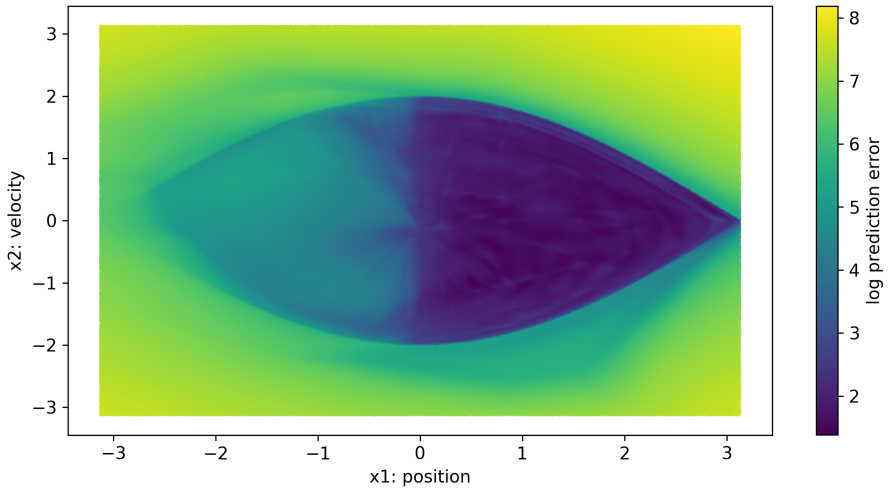


Figure 6: The same plot as Figure 5, but with the network trained on the part of the $H < 1$ region where x_1 is positive, i.e., the right half-oval. Clearly, the network poorly extrapolates to the left half, where the prediction error is much higher. Extrapolation to the region outside of $H < 1$ is even more prone to high prediction errors, as the contour of the lower-energy oval can clearly be identified in the left half.

the decoded Koopman layer output is subtracted from the actual trajectory. Lusch et al. [2018] employ this term in their loss function to "achieve linear dynamics". However, linear dynamics are already ensured by the network architecture, explicitly by using a linear activation function and no bias in the Koopman layer. When this loss term is added, the mean prediction error in the setting discussed in the first paragraph increases from 90 to 95.

Parametrised Koopman operator. So far, the network architecture has been left unchanged. In a last step, I tried to make the architecture resemble that of Lusch et al. [2018] more closely by making the Koopman layer as well as the encoding space two-dimensional. Furthermore, the Koopman layer receives a third one-number input, generated by an auxiliary network with one hidden layer of width 140 as in the paper. A direct use of this value only as a matrix entry was not possible in the Keras library.

Figure 8 shows the result of a 30-epoch training run with the auxiliary network on training data with zero initial velocity (on the x-axis). Similarly to Figure 5, predictions are better in the training data region even though the difference is not as marked. The overall prediction error decreases from 80 to 65 for the average trajectory in the $H < 1$ region.

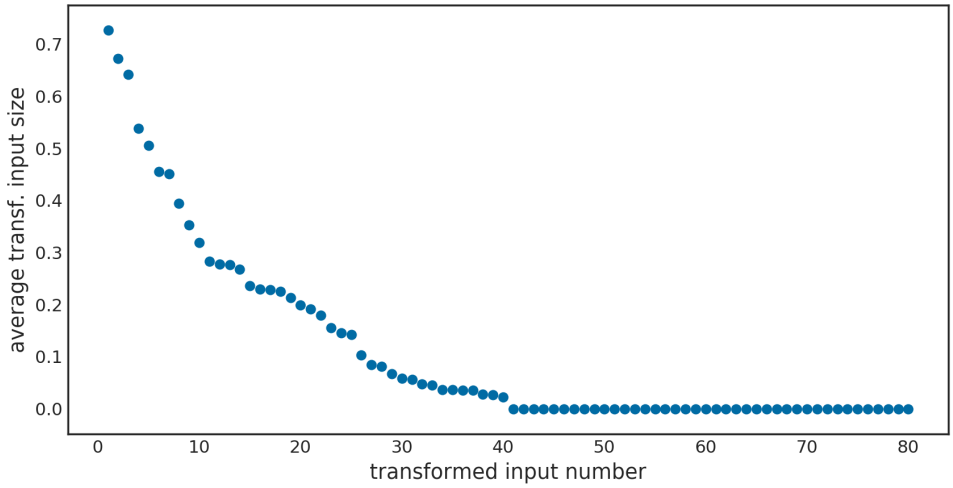


Figure 7: The average value of the encoded input vector entries sorted in descending order. The encoding with the 80-dimensional dictionary $\mathbf{\Psi}$ outputs vectors that are half-empty. Therefore, the eigenvalues shown in Fig. 4 may differ from those expected for the Koopman operator describing the evolution of the system.

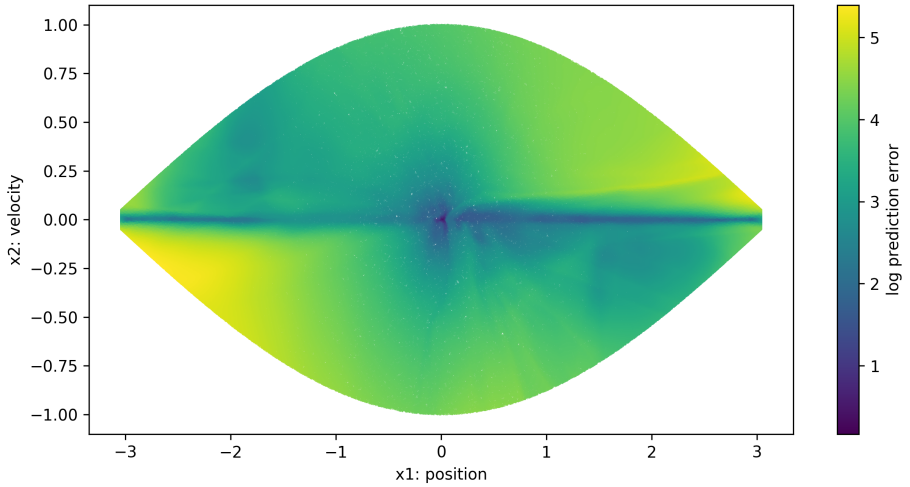


Figure 8: With an auxiliary network which parametrises the Koopman operator the architecture seems to "smoothen out" the difference in prediction accuracy seen in Figure 5 between training and non-training regions of phase space. The average prediction error per trajectory drops from 80 to 65.

3 Conclusion

In the above, I review the development of Koopman analysis with neural networks. This development started with Dynamic Mode Decomposition (DMD), which looks for patterns in fluid flows evolving linearly in time. DMD was extended to EDMD, allowing for a nonlinear transformation of state space variables. Employing autoencoder neural networks for the choice of dictionary functions in EDMD is the final step in the development to the technique employed by Lusch et al. [2018].

Koopman eigenfunctions and eigenvalues are derived analytically. A neural network architecture similar to the one used by Lusch et al. [2018] is implemented. In a first step, a simpler version is implemented, lacking an auxiliary network and one of the loss terms. Nevertheless, it performs very well predicting trajectories of the nonlinear pendulum to great accuracy after a short training period. It also clearly shows nonlinear effects (increase in the pendulum period) even though its time-evolution layer is linear.

However, accuracy drops dramatically for trajectories starting outside the training data region of state space. Prediction accuracy drops further when trajectories describe the pendulum "swinging through" its highest point (instead of swinging back and forth, with the velocity changing sign).

A similar behaviour is observed for a more sophisticated version of the network, using both an additional loss term taking differences in latent variable space and an auxiliary network to parametrise the Koopman operator. However, this network has a slightly lower average prediction error and the drop in prediction accuracy outside the training data region is less pronounced.

The results raise some doubts about the use cases of Koopman analysis with neural networks. On the one hand, they show that it is possible to train a neural network to transform state space data to variables where a linear time evolution governs the trajectory. But on the other hand, it seems that generalisations from the training data to different regions of state space are prone to large errors. This means that for every trajectory the network should be able to predict, it has to be trained with a sufficient amount of sufficiently nearby data. Essentially, the network has to be seen as a machine doing *interpolation* and not *extrapolation*.

Various possibilities are open for further research: The above statement about sufficient training data could be made quantitative by exploring the impact of the sparsity of training trajectories, as well as by further quantifying the prediction errors for different extrapolation settings. Furthermore, to approach the implementation of Lusch et al. [2018], the network could be implemented in TensorFlow to parametrise the Koopman operator directly, using an auxiliary network to output the weights in the Koopman layer. Then, it

would also be possible to enforce the eigenvalues' lying on the unit circle.

It would also be interesting to try different dimensionalities, effectively truncating the Koopman operator earlier, as the results suggest that not all of the dictionary's dimensions are used (many entries are zero in the encoding). However, this could also be linked to the so-called "dying ReLU problem": When the neuron is updated with a large negative bias, all inputs will be mapped to zero. This means that different inputs always give the same output and the neuron weights and biases stops being updated. To avoid this, a sigmoid or other activation function with non-vanishing gradient could be investigated.

From a mathematical perspective, the results may not be surprising: The universal approximation theorem states that neural networks can approximate arbitrary continuous functions *with compact support*. In the above results, it has been shown that the neural network employed here actually does learn functions which are much more accurate on the support of the state space region the network was trained on. The approximation is good only on the training data support, which is a ubiquitous problem in machine learning.

Bibliography

Hassan Arbabi. Introduction to Koopman operator theory of dynamical systems. *pre-print*, 2018. `https://sites.engineering.ucsb.edu/~harbabi/research/KoopmanIntro.pdf`.

Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

François Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Anonymous essay author. Code for essay neural networks to linearise nonlinear dynamics. `https://github.com/ManshaP/KoopmanNets`, 2020.

Nithin Govindarajan, Hassan Arbabi, Louis van Blargian, Timothy Matchen, Emma Tegling, et al. An operator-theoretic viewpoint to non-smooth dynamical systems: Koopman analysis of a hybrid pendulum. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 6477–6484. IEEE, 2016.

Eurika Kaiser, Bernd R Noack, Laurent Cordier, Andreas Spohn, Marc Segond, Markus Abel, Guillaume Daviller, Jan Östh, Siniša Krajnović, and Robert K Niven. Cluster-based reduced-order modelling of a mixing layer. *Journal of Fluid Mechanics*, 754: 365–414, 2014.

Bernard O Koopman. Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences of the United States of America*, 17(5):315, 1931.

Milan Korda and Igor Mezić. On convergence of extended dynamic mode decomposition to the koopman operator. *Journal of Nonlinear Science*, 28(2):687–710, 2018.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Qianxiao Li, Felix Dietrich, Erik M Bollt, and Ioannis G Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):4950, 2018.

Igor Mezić. Analysis of fluid flows via spectral properties of the koopman operator. *Annual Review of Fluid Mechanics*, 45:357–378, 2013.

Bernd R Noack, Konstantin Afanasiev, Marek Morzyński, Gilead Tadmor, and Frank Thiele. A hierarchy of low-dimensional models for the transient and post-transient cylinder wake. *Journal of Fluid Mechanics*, 497:335–363, 2003.

Samuel E Otto and Clarence W Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.

Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. In *Advances in Neural Information Processing Systems*, pages 1130–1140, 2017.

Jonathan H Tu, Clarence W Rowley, Dirk M Luchtenburg, Steven L Brunton, and J Nathan Kutz. On dynamic mode decomposition: Theory and applications. *arXiv preprint arXiv:1312.0041*, 2013.

Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, pages 4832–4839. IEEE, 2019.