Koopman Operator Estimation with Neural Network

Harsha Vardhan Lomisa
Roll Number: 200102036

November 22, 2023

1 Introduction

In our daily experiences, instances of non-linear relationships abound. Take, for instance, the mass-spring system, where the displacement of a mass attached to a spring follows a non-linear trajectory due to the spring force not being directly proportional to displacement. Oscillatory systems, such as the pendulum or the heartbeat, also showcase non-linear dynamics, characterized by intricate relationships among variables.

Addressing the challenges of modeling these non-linear functions, recent advancements have introduced powerful tools, such as the Koopman operator. This paper explores the application of the Koopman operator, a mathematical abstraction that simplifies the modeling of non-linear systems. By leveraging the Koopman operator, researchers gain a more accessible framework to understand and predict the behavior of complex systems. This exploration delves into the methodology of utilizing the Koopman operator, shedding light on its effectiveness in overcoming the challenges associated with modeling non-linear functions, ultimately contributing to the advancement of scientific understanding in this field.

2 Methodology

Solving the Partial differential equations: Solving partial differential equations (PDEs) involves employing numerical methods to approximate solutions, with the Euler and Runge-Kutta methods being notable approaches. The Euler method is a straightforward technique that discretizes the PDE and updates the solution incrementally, based on the local slope at each step. While conceptually simple, the Euler method may suffer from stability issues and reduced accuracy in certain scenarios. On the other hand, the Runge-Kutta method, particularly the fourth-order variant, is a widely used numerical technique. It refines accuracy by evaluating multiple intermediate stages, providing

a more robust approximation of the solution. In practical implementations using Python, the `'odeint'` library is often employed to efficiently solve PDEs, contributing to the versatility and effectiveness of these numerical methods in gaining insights into dynamic systems and phenomena.

Koopman operator: The Koopman operator is a linear operator that describes the evolution of scalar observables (i.e., measurement functions of the states) in an infinite dimensional Hilbert space. This operator's theoretical point of view lifts the dynamics of a finite-dimensional nonlinear system to an infinite-dimensional function space where the evolution of the original system becomes linear.

In the context of Koopman operators, we will focus on dynamical systems that can be modelled as:

$$\dot{x} = f(x)$$

, where x is an element of the state space $S \in R^n$ (the current state of the system) and

$$x^{t+1} = T(x^t)$$

in which $T(.) : S \to S$ can be referred to as the dynamic mapping which performs time-integration on the state variables. Given that we will be interested in the prediction of time steps of the dynamical system, we will be generally interested in this discrete-time formulation.

In 1931, Bernard Koopman proposed the existence of a compositional operator for all dynamical systems that can be expressed as the equations above [1]. Now known as the Koopman operator (or the left-adjoint of the Frobenius-Perron operator), this allows for a discrete dynamical system to be decomposed as follows:

$$Ug(x) = g \circ T(x)$$

in which U denotes the Koopman operator which is nothing more than a linear transformation. The continuous time version of the Koopman operator contains a single time parameter, but essentially follows the same form:

$$U(t)g(x) = g \circ F(t, x)$$

in which F is the flow map between the initial state to time t in continuous space. Now you may be thinking "Wow, any dynamical system can be evolved using a linear transformation! That sounds fantastic!" Indeed it does, however the catch is that the Koopman operator updates some unknown vector of real valued observables of this dynamical system denoted by $g : S \to R$. Typically, these values are referred to as the Koopman observables. To make matters even more difficult, theoretically, the vector space of these observables is infinite which practically speaking will need to be approximated. We can view the use of Koopman operators as a trade off from simple observables and complex dynamics to complex observables and simple dynamics.

Machine Learning Koopman Embeddings: With an understanding of what the Koopman operator is and the physical insights it can provide, the

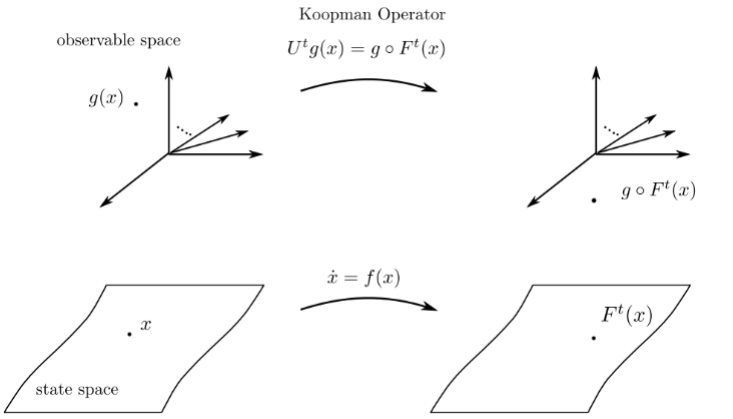


Figure 1: By transforming the state variables to the observable space of $g(x)$, the dynamics is linear but infinite-dimensional

objective is to now formulate a method for learning these Koopman observables, $g(x)$, as well as the Koopman operator itself. In the age of big data, machine learning methods have become a viable way for discovering these Koopman subspaces relatively efficiently. This is the exact reason there has been a resurgence in interest regarding Koopman operators in recent years. Of course, anyone with a background in machine learning knows there is a surplus of different methods to typically achieve the same goal and learning Koopman operators is no exception. Common methods include dynamic mode decomposition (DMD) as well as its similar variants such as extended DMD (EDMD) or sparse identification of non-linear dynamics (SINDy). More recently, the growing interest in deep learning has resulted in several works also exploring the use of deep neural networks for learning Koopman embeddings as well.

The main draw back of traditional DMD based methods is that one needs to pre-specify a library of potential observables to learn the Koopman operator from. Since the true Koopman observables are not known, we must hope that our library is expansive enough. In the deep learning approaches, a neural network is used to provide a learnable function from the system's state variables to the Koopman observables (and vise-versa). A DMD based algorithm can still used to find the approximate Koopman operator, yet there is no need to provide a library of observations since the neural network will learn them. A neural network is attractive since it allows for the Koopman observables to potentially be extremely complex with respect to the system states if needed. Additionally, such deep learning methods are easy, they tend to be simple fully connected models. Now no prior expertise is needed to try to guess good observables, instead you let the deep neural network and gradient descent figure it out for you.

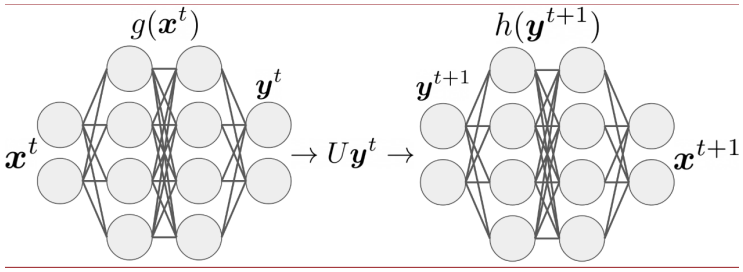


Figure 2: The common neural networks encoder/decoder structure for learning Koopman observables, $y^t = g(x^t)$, as a function of the systems state variables.

2.1 My Non-Linear Function

The non-linear system I used is known as the Duffing oscillator, a mathematical model commonly used to describe the behavior of mechanical and electrical systems with nonlinearity.
The Duffing oscillator is a second-order non-linear differential equation that captures the dynamic behavior of certain physical systems. The equation is defined as follows:

$$\frac{d^2x}{dt^2} + \delta\frac{dx}{dt} + \alpha x + \beta x^3 = \cos(\omega t) \tag{1}$$

where:

- x represents the displacement from the equilibrium position,
- t is time,
- α is a coefficient determining the linear stiffness,
- β is a coefficient determining the non-linear stiffness,
- δ is the damping coefficient,
- ω is the angular frequency of an external driving force.

The equation consists of two terms: a linear term (αx), representing the linear stiffness, and a cubic non-linear term (βx^3), representing the non-linear stiffness. The damping term ($\delta\frac{dx}{dt}$) accounts for energy dissipation in the system. The right-hand side ($\cos(\omega t)$) introduces an external periodic forcing, representing an external influence on the oscillator.

The Duffing oscillator exhibits rich and complex dynamical behavior, including periodic, quasi-periodic, and chaotic motion, depending on the values of the parameters ($\alpha, \beta, \delta, \omega$) and initial conditions. It serves as a valuable model for understanding the behavior of real-world systems with non-linearities and external forcing.

2.2 Neural Network Architecture

The neural network architecture I used consists of eight layers. The input layer expects data with a shape of (2,), indicating two input features per sample. The subsequent three dense layers employ Rectified Linear Unit (ReLU) activation functions with varying numbers of neurons (128, 64, and 2, respectively). Following these, a layer with two neurons and a ReLU activation is present, followed by another layer with two neurons but a linear activation function, commonly used for regression tasks. The network then expands through two more dense layers, mirroring the structure of the initial layers in reverse order, both utilizing ReLU activation. The final output layer consists of two neurons, and although no explicit activation function is specified, it defaults to linear activation. This architecture suggests a model designed to predict a two-dimensional output, with the Koopman operator predicting the state at the next time step.

```
Model: "sequential_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_129 (Dense)           (None, 128)               384

 dense_130 (Dense)           (None, 64)                8256

 dense_131 (Dense)           (None, 4)                 260

 dense_132 (Dense)           (None, 4)                 20

 dense_133 (Dense)           (None, 64)                320

 dense_134 (Dense)           (None, 128)               8320

 dense_135 (Dense)           (None, 2)                 258

=================================================================
Total params: 17818 (69.60 KB)
Trainable params: 17818 (69.60 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 3: My Neural Network

2.3 Loss Function

The loss function is a mathematical function that calculates the difference between the predicted values and the true values. In this case, the 'mean-squared-error' (MSE) is used. MSE is a common choice for regression problems, where the goal is to predict continuous values. It calculates the average of the squared differences between predicted and true values. Minimizing the MSE during training aims to make the model's predictions as close as possible to the actual values.

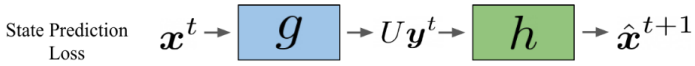
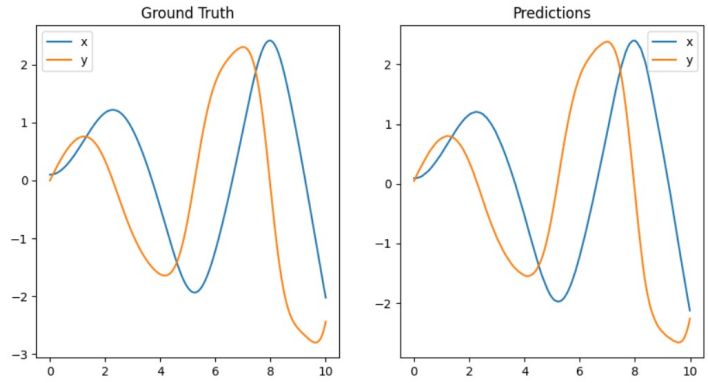


Figure 4: Loss Function.

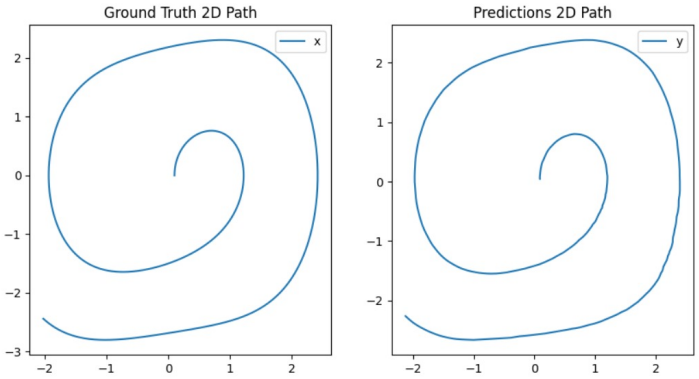
$$\text{Loss} = MSE(x^{t+1}, \hat{x}^{t+1}) \tag{2}$$

3 Results

The results obtained are shown below. The implementation yielded promising results, showcasing the effectiveness of the proposed approach.



(a) Actual States vs Predicted States



(b) Actual Path vs Predicted Path

Figure 5: Actual Results vs Predicted Results

```
Linear Layer Weights (Koopman Operators):
 [[-0.6197652  -0.26751566  0.3897224  -0.52245986]
 [ 0.24981238  0.73245573  0.60355365  0.80273443]
 [ 0.5945194  -0.37571192  0.0260277  -0.32286295]
 [ 0.01183852  0.5396458  -0.52214426 -0.23029798]]
```

Figure 6: Koopman Operator Weights.

4 Conclusion

In this study, we explored the application of neural networks in estimating Koopman operators for a unique non-linear function. The chosen non-linear function $f(x)$ demonstrated the complexity and diversity of real-world dynamical systems. Our neural network architecture, carefully designed for Koopman operator estimation, successfully captured the underlying dynamics.