DEEP LEARNING MODELS FOR DYNAMICAL SYSTEMS

Alan Williams* *and Imoleayo Abel**

*Dept. of Mechanical and Aerospace Engineering, University of California San Diego.

1. INTRODUCTION

Nonlinear dynamical systems abound in nature and very rarely do real-life systems exhibit purely linear dynamics. However, existing results on the analysis nonlinear systems pale in comparison to the large body of work on linear systems. As a result, it is often desirable to find representations of nonlinear systems that render them amenable to the sophisticated tool-set available for linear system analysis and control.

Koopman Operator Theory provides an ideal framework for developing ways of understanding linear representations of nonlinear systems. The theory shows that all nonlinear systems amend themselves to a description of linear dynamics given the correct state transformation, although the transformation may be very large or infinite in dimension [1] [2]. In [3], Dynamic Mode Decomposition (DMD) was introduced as a method to learn an estimate of the Koopman operator, and was extended using a method called Extended-DMD [4]. These methods involve first measuring a set of observables (functions of the system state) in time, and then solving for the best fit linear operator describing the dynamics. [5] later discovered that using an autoencoder style network, a lower-dimensional, more manageable representation of the system state may also be learned in addition to the linear operator describing the dynamics. More recently, [6] elegantly combined the aforementioned works, in which they learned some transformation to a latent space (using an autoencoder style neural network), and the linear dynamics operator, from sampled data of the nonlinear system. We try to recreate the results from [6] in the first part of this project.

The utility of the Koopman operator lies in the mechanism of a state transformation which leads to a simpler representation of the system dynamics. This idea sounds similar to a powerful technique in the field of Control Theory called "feedback linearization" (FBL). FBL is a technique in which we can represent the system equations in a linear form with a state transformation (as in Koopman Operator Theory) and an input transformation. For control theorists this is useful because it allows one to use powerful methods in linear control to develop feedback strategies for inherently nonlinear systems. If we suspect a system can be feedback linearized but have no knowledge of the system model, we attempt to identify a model of the system and yield a form of the model which may readily be controlled with feedback. We attempt to accomplish this task by learning the state and input transformations from data. The details of FBL theory is extensive and left for the reader [7] [8] [9]. The theory also extends to systems with multiple inputs and systems which can only be partially feedback linearized.

Notation: We denote a vector $\mathbf{x} \in \mathbb{R}^n$ in lowercase bold font and it's components $x_1, \ldots, x_n$ in regular fonts. We use $\mathbf{x_t}$ to denote the full n-dimensional vector at time t. Therefore, x_4 represents the fourth component of vector $\mathbf{x}$, while $\mathbf{x_4}$ represents the full n-dimensional vector at time $t = 4$. For denoting the j-th component of $\mathbf{x_t}$, we write $x_j(t)$. In addition, $\mathbf{x_{1:m}}$ denotes a set of m vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_m} \in \mathbb{R}^n$ constituting a trajectory from time $t = 1$ to $t = m$. Lastly, for matrices, we use uppercase bold font $\mathbf{A}$.

2. PROBLEM DESCRIPTION

The first dynamical system we consider is the following 2-dimensional nonlinear system from [6],

$$\begin{aligned} \dot{x}_1 &= \mu x_1 \\ \dot{x}_2 &= \lambda(x_2 - x_1^2) \end{aligned} \tag{1}$$

We study this system in an effort to reproduce the results in [6], and also because it is one for which there is a known finite-dimensional state transformation that transforms the system into a linear system. In particular, using the transformation

$$\mathbf{y} := \varphi(\mathbf{x}) = \begin{pmatrix} x_1 \\ x_2 \\ x_1^2 \end{pmatrix}, \tag{2}$$

the nonlinear dynamics in (1) is transformed into the linear system

$$\begin{aligned} \dot{\varphi}(\mathbf{x}) &= \begin{bmatrix} \mu & 0 & 0 \\ 0 & \lambda & -\lambda \\ 0 & 0 & 2\mu \end{bmatrix} \varphi(\mathbf{x}) \\ \dot{\mathbf{y}} &= \mathbf{K}\mathbf{y} \end{aligned} \tag{3}$$

While the example system above is one for which a finite-dimensional transformation exists, some nonlinear systems require an infinite-dimensional transformation to represent them linearly. In such cases, a data-based approach would learn the "best" (in terms of a specified loss function) finite-dimensional transformation in the region of the state space the data was generated from.

For this first system, described henceforth as the "Koopman problem" (KPM), our goal is to design an autoencoder network to learn the nonlinear state-transformation $x \mapsto \varphi(x)$, and the linear Koopman operator $\mathbf{K}$ that renders (1) linear.

The theory of FBL gives us a body of work which is analogous to Koopman Operator Theory. We will be using the following example to demonstrate FBL and will use the same system in our experiment. To extend the results from [6], we consider the following nonlinear system with input u,

$$\begin{aligned} \dot{x}_1 &= \sin(x_2) \\ \dot{x}_2 &= -x_1^2 + u \end{aligned} \tag{4}$$

Using state transform $\mathbf{x} \mapsto T(\mathbf{x})$ and input transform $u \mapsto g(u; \mathbf{x})$ defined below,

$$\begin{aligned} \mathbf{z} &:= T(\mathbf{x}) = \begin{pmatrix} x_1 \\ \sin x_2 \end{pmatrix} \\ \mathbf{v} &:= g(\mathbf{u}; \mathbf{x}) = u \cos x_2 - x_1^2 \end{aligned} \tag{5}$$

we can transform the system into new state and input spaces $\mathcal{Z}$ and $\mathcal{V}$ respectively where we get the following linear dynamics in $\mathbf{z}$ and $\mathbf{v}$

$$\dot{\mathbf{z}} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \mathbf{z} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \mathbf{v} \tag{6}$$

Now the model is in a suitable form for a variety of linear control techniques. This is an example of full state feedback linearization. Our question for this second system is therefore the following: using data sampled from the original system, can we learn the transformations T and g?

Note: we utilize the equivalent, discrete version of (6) listed below for a zero-order hold input over a sample time of $dt = 0.1$.

$$\mathbf{z_{t+1}} = \underbrace{\begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}}_{\mathbf{A}} \mathbf{z_t} + \underbrace{\begin{bmatrix} 0.005 \\ 0.1 \end{bmatrix}}_{\mathbf{B}} \mathbf{v}_t \tag{7}$$

For both the KPM and FBL problems, the transformations φ, T, and g are required to be invertible.

2.1. Loss function

For KPM, we utilize the following loss function (akin to [6]) to characterize the quality of the transformation φ and operator $\mathbf{K}$ learned from the data

$$\mathcal{L}_{total} = \alpha_1 \mathcal{L}_{recon} + \alpha_2 \mathcal{L}_{pred} + \alpha_3 \mathcal{L}_{lin} + \alpha_4 \|\mathbf{W}\|_2^2 \tag{8}$$

where

$$\mathcal{L}_{recon} = \left\| \mathbf{x_1} - \varphi^{-1}(\varphi(\mathbf{x_1})) \right\|_{MSE} \tag{9}$$

$$\mathcal{L}_{pred} = \frac{1}{S_p} \sum_{m=1}^{S_p} \left\| \mathbf{x_{m+1}} - \varphi^{-1}(\mathbf{K}^m \varphi(\mathbf{x_1})) \right\|_{MSE} \tag{10}$$

$$\mathcal{L}_{lin} = \frac{1}{S_p} \sum_{m=1}^{S_p} \| \varphi(\mathbf{x_{m+1}}) - \mathbf{K}^m \varphi(\mathbf{x_1}) \|_{MSE} \tag{11}$$

The reconstruction loss $\mathcal{L}_{recon}$ simply measures the quality of the state transformation learned as well as it's inverse. This is similar to what an autoencoder loss function would be. Secondly, the prediction loss $\mathcal{L}_{pred}$ encapsulates the ability of the learned transformation and linear operator to successfully predict future states of the system over a range of time horizons. Lastly, the linearization loss $\mathcal{L}_{lin}$ attempts to isolate the effect of the linear operator $\mathbf{K}$ in moving the system forward in time in the transformed linear space. The variable S_p represents the largest time-horizon prediction used for learning.

For the FBL problem, we use a similar loss function but without the linearization loss $\mathcal{L}_{lin}$. This is because the linear system matrices $\mathbf{A}$ and $\mathbf{B}$ are given explicitly if the correct transformation is found, and therefore do not need to be learned. Specifically, the loss function for FBL is given as

$$\mathcal{L}_{total} = \beta_1 \mathcal{L}_{recon} + \beta_2 \mathcal{L}_{pred} + \beta_3 \|\mathbf{W}\|_2^2 \tag{12}$$

where

$$\begin{aligned} \mathcal{L}_{recon} = & \left\| \mathbf{x_1} - T^{-1}(T(\mathbf{x_1})) \right\|_{MSE} \\ & + \left\| \mathbf{u_1} - g^{-1}(g(\mathbf{u_1}; \mathbf{x_1}); \mathbf{x_1}) \right\|_{MSE} \end{aligned} \tag{13}$$

$$\mathcal{L}_{pred} = \frac{1}{S_p} \sum_{m=1}^{S_p} \| \mathbf{x_{m+1}} - P_m(\mathbf{x_1}, \mathbf{u_{1:m}}) \|_{MSE} \tag{14}$$

Here, P_m is the m-step ahead prediction given the initial condition $\mathbf{x_1}$ and inputs up until the time m. For example, for $m = 2$, we can represent mathematically $P_m(\mathbf{x_1}, \mathbf{u_{1:m}})$ as,

$$P_2(\mathbf{x_1}, \mathbf{u_{1:2}}) = T^{-1}(\mathbf{A}T(\hat{\mathbf{x}}_\mathbf{2}) + \mathbf{B}g(\mathbf{u_2}; \hat{\mathbf{x}}_\mathbf{2})) \tag{15}$$

where $\hat{\mathbf{x}}_\mathbf{2}$ is,

$$\hat{\mathbf{x}}_\mathbf{2} = T^{-1}(\mathbf{A}T(\mathbf{x_1}) + \mathbf{B}g(\mathbf{u_1}; \mathbf{x_1})) = P_1(\mathbf{x_1}, \mathbf{u_1}) \tag{16}$$

We note here that we have to include $\mathbf{A}$, $\mathbf{B}$, in the network which are based upon the assumption that our system is feedback linearizable. These consist of weights that are held constant throughout training, to enforce this assumption.

3. NETWORK ARCHITECTURE

For the KPM problem, the network consisted of three submodels: an encoder for learning φ, a decoder for learning φ^{-1} and a dense fully connected network for learning $\mathbf{K}$.

Using these sub-models, the full network was designed with different branches consisting of different number of passes through the sub-models. Specifically, the full network consists of a $2 \times (S_p + 1)$ input-layer representing a state trajectory $\mathbf{x_{1:(S_p+1)}}$ where each $\mathbf{x_j} \in \mathbb{R}^2$. The output of the full network consists of the following three sets of nodes corresponding to the three losses.

(i) 2 nodes representing the reconstruction of the initial condition $\mathbf{x_1}$ of the input trajectory i.e. $\varphi^{-1}(\varphi(\mathbf{x_1}))$. When training, the label for this pair of output nodes is the state $\mathbf{x_1}$ in the input trajectory.

(ii) $2S_p$ nodes i.e. S_p pairs of nodes, with the m-th pair representing the prediction of the state $\mathbf{x_{m+1}}$ in the input trajectory i.e. $\varphi^{-1}(\mathbf{K}^m\varphi(\mathbf{x_1}))$. When training, the label for the m-th pair of output nodes is the state $\mathbf{x_{m+1}}$ in the input trajectory.

(iii) $2S_p$ nodes i.e. S_p pairs of nodes, with the m-th pair representing the difference between the latent space representation of state $\mathbf{x_{m+1}}$ in the input trajectory, and the latent-space representation of the m-step prediction of the initial condition $\mathbf{x_1}$ i.e. $\varphi(\mathbf{x_{m+1}}) - \mathbf{K}^m\varphi(\mathbf{x_1})$. When training, the label for all $2S_p$ output nodes is 0

In all, the full network for the KPM problem has an output layer with $4S_p + 2$ nodes. This full network can be thought of as $2S_p + 1$ separate sub-networks (each with 2 output nodes) that represent different number of passes through the φ, $\mathbf{K}$, and φ^{-1} sub-models, while all sharing the same weights from the sub-models. This weight-sharing capability is enabled by the Keras functional API in Tensorflow.

The network architecture for the FBL problem was designed in a similar manner with sub-models T, T^{-1}, g, and g^{-1} acting as building blocks. Here, the full network has a $3 \times (S_p + 1)$ input layer corresponding to a state trajectory $\mathbf{x_{1:(S_p+1)}}$ and applied inputs $\mathbf{u_{1:(S_p+1)}}$ with each $\mathbf{u_j} \in \mathbb{R}$. In addition, the full network has a total of $2S_p + 3$ output nodes: 2 nodes for the reconstruction of the initial state $\mathbf{x_1}$, 1 node for the reconstruction of the initial input $\mathbf{u_1}$, and $2S_p$ nodes for the state predictions.

4. METHODS

4.1. Data generation

For the KPM problem, we use $\mu = -0.05$, $\lambda = -1$, and $S_p = 50$ with $dt = 0.02$ that is, one trajectory comprises of state values at times $0, 0.02, 0.04, \ldots, 1.02$ as in [6]. We use initial condition $x_1(0), x_2(0)$ chosen randomly in the range $[-0.5, 0.5]$ for each trajectory. In all, we used a dataset with 13,000 trajectories with a 10000-2000-1000 split for training, validation, and testing. We generated this dataset using an explicit solution of the system (3) in the transformed linear $\mathbf{y}$-space because the inverse transformation (2) from $\mathbf{y}$ to $\mathbf{x}$ is identity for the first two components. Specifically, we generated x_1, x_2 data as follows

$$\begin{aligned} x_1(t) &= y_1(t) = y_1(0)e^{\mu t} \\ x_2(t) &= y_2(t) = \left(y_2(0) + \frac{\lambda}{2\mu - \lambda}y_3(0)\right)e^{\lambda t} \\ &\qquad - \frac{\lambda}{2\mu - \lambda}y_3(0)e^{2\mu t} \end{aligned} \tag{17}$$

For FBL, we generated 5500 trajectories with an 4500-500-500 split. We used a Simulink model to simulate the nonlinear equations (4), generating random input (zero order hold) between -5 and 5 throughout $S_p = 21$ time steps with $dt = 0.1$. Trajectories were only kept for training if states remained within -0.5π and 0.5π: the invertible region of the theoretical transformation.

4.2. Network and training parameters

For the KPM problem, we achieved the best results with the following network parameters. The φ, φ^{-1}, and $\mathbf{K}$ sub-models have 2 dense same-width hidden layers of widths 120, 120, and 80 respectively. The loss weights were $\alpha_1 = 5, \alpha_2 = 3, \alpha_3 = 0.1, \alpha_4 = 10^{-10}$. The network was trained for 100 epochs. For FBL, the functions T, T^{-1}, g, and g^{-1} were modeled with 2 dense hidden layers 120 nodes wide. During training we achieved the best results with 150 epochs and loss weights $\beta_1 = 1, \beta_2 = 5, \beta_3 = 0$. We did not experience over-fitting for the FBL problem so regularization was not needed.

5. RESULTS AND CONCLUSION

As seen in Fig. 1, the transformation learned for the KPM problem mirrors the expected theoretical results. While these transformations do not match exactly, the reconstruction of the states were as good as perfect. This is likely due to the fact that we do not directly penalize the latent space representation, and as such the system settles on weight that minimize terms in the loss functions. In Fig. 2, we see that the network is able to predict trajectory data appreciably well from a given initial condition.

Fig. 3 and Fig. 4 describes T_2 and the input transformation g. The input transformation was not as expected because the theoretical and generated plots do not align when different points in the state spaced are held fixed - although the general shape is as expected. The network generated the function T_2 accurately, but lacks the precision to converge to the sine function. T_1 that the network generated and the reconstruction of input and state are excellent, but not shown. Prediction power in Fig. 5 is satisfactory.

One difficulty we came across in both the KPM and FBL problem was that training seemed sensitive to weight re-initialization and some hyperparameters. This may just be the nature of the ML tuning process or the design of the loss functions (as we did not include one of the loss terms from [6]), or a combination of both.

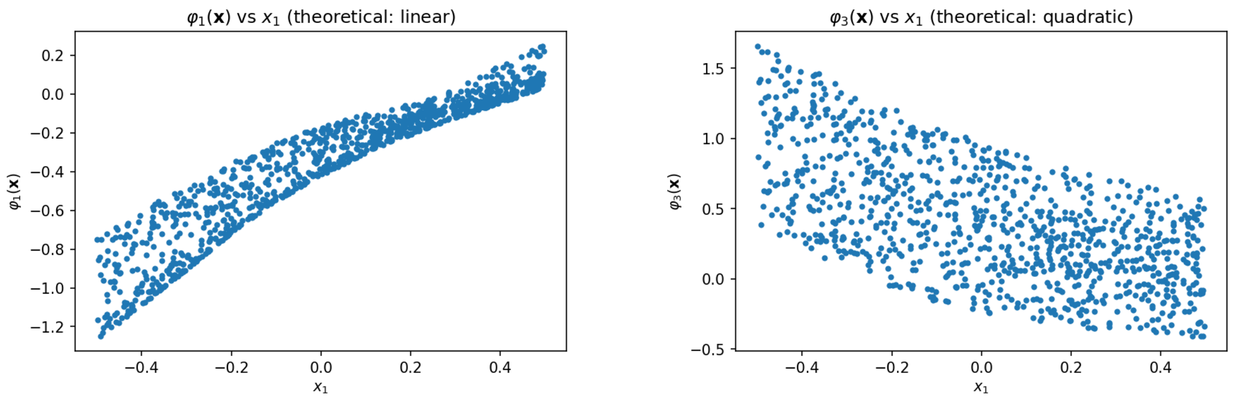

Fig. 1. KPM latent space representations of $\varphi_1(\mathbf{x})$ and $\varphi_3(\mathbf{x})$ which theoretically should be x_1 and x_1^2 respectively.

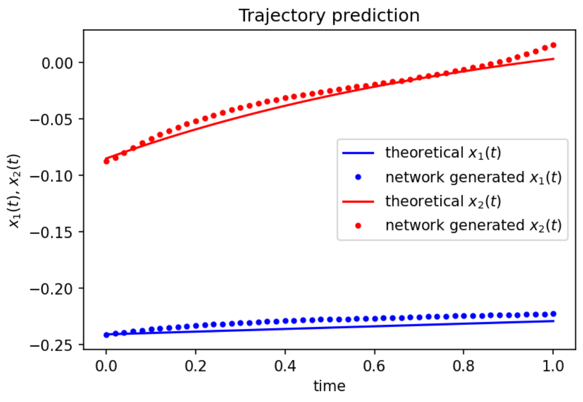

Fig. 2. KPM trajectory prediction for 51 time-steps.

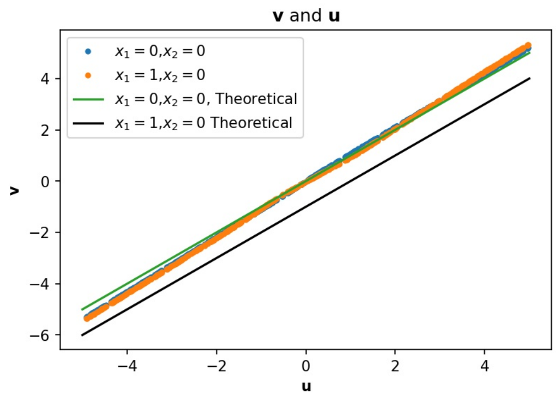

Fig. 4. $g(\mathbf{u}; \mathbf{x})$ drawn randomly from the test set.

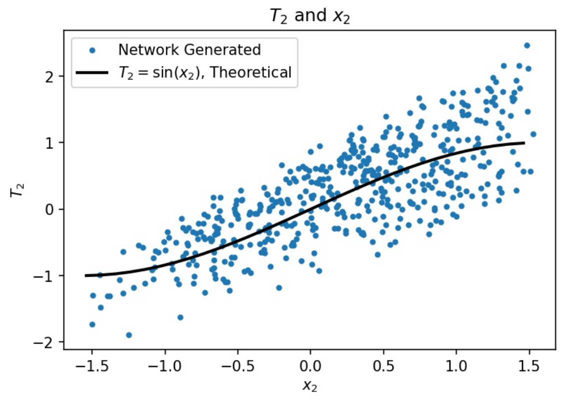

Fig. 3. T_2 drawn randomly from the test set, with varying x_1 values.

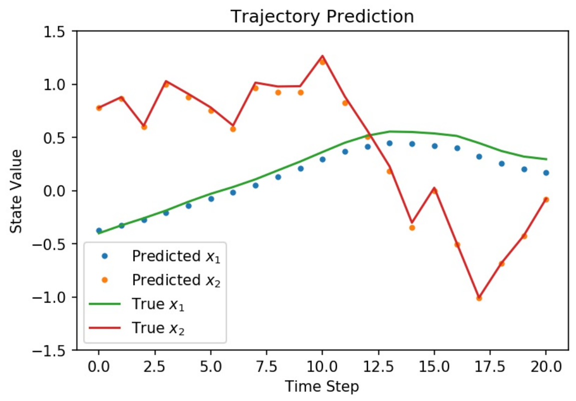

Fig. 5. FBL trajectory prediction over 21 time steps.

Given that the FBL problem we chose to tackle had not been previously considered (*to the best of our knowledge*), we were impressed that we achieved some semblance of the theoretical results from FBL theory. The functions that were generated trend in the expected way, and this paper may be considered as a preliminary to future work. Future goals include designing a learning framework that may easily generalize to systems of the same dimension without much tuning on the part of the engineer or scientist.

6. CONTRIBUTION

Imoleayo Abel was responsible for the KPM portion of the project that attempt to reproduce the result of [6]. Alan Williams was responsible for the FBL portion of the project that extends the results of [6] to nonlinear system with inputs.

7. REFERENCES

[1] B. O. Koopman. Hamiltonian systems and transformation in hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318, 1931.

[2] B. O. Koopman and J. V. Neumann. Dynamical systems of continuous spectra. *Proceedings of the National Academy of Sciences of the United States of America*, 18(3):255–263, Mar 1932.

[3] J. Nathan Kutz, Steven L. Brunton, Bingni W. Brunton, and Joshua L. Proctor. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[4] 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, Dec 2015.

[5] 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.

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

[7] Jorge Cortes. Lecture 3: Feedback linearization of MIMO systems. In *MAE 281B: Nonlinear Control*, pages 1–11. UCSD, San Diego, 2019.

[8] Alberto Isidori. Elementary theory of nonlinear feedback for single-input single-output systems. In *Nonlinear Control Systems*, pages 137–217. Springer London, London, 1995.

[9] Hassan K Khalil. *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ, 2002.

8. CRITIQUE RESPONSES

8.1. Critique by Group 35

Alan and Imoleayo did a great job in introduction of the basic mathematical derivation in linearization. They provided a very solid background for their readers. Also, they introduced a new ML model, Koopman operator and their loss function. Their final data showed that they achieved their expected results. Also, their codes are quite sophisticated and run successfully. Overall, we believe that this project has a lot of mathematical derivation work to finish and they managed to do that. As for their presentation, we believe that they delivered a very clear background and provided very good information to the viewers.

- One thing to critique, data Size of 5000 seems too small to train a good machine learning model, maybe you should think try generate more data to fit a better model.

- **Response**: For the Koopman problem, we increased the data size to 13,000. For feedback linearization, we kept the data-size at 5000 which achieved sufficiently good results, and is comparable to the data-size used in [6]

8.2. Critique by Group 69

Thought that you did a good job explaining a few chapters worth of control theory in 5 minutes. Can't say I understood all of it, but that was on me and not on you as the presenter. It is always interesting seeing ML models that still maintain some semblance to a linear system but mix in some deep learning elements as part of the system. The Multitask learning part of your architecture appears well designed and indicated your domain knowledge. The graphs of performance also did a good job of telling the story of how the model performed beyond the loss function.

Critiques/Improvements:

- An ablation study of how the model performed with and without the various components of the loss function would have been nice.

- **Response**: We believe the loss functions used were necessary and in fact, we dropped one loss function considered in the original paper [6] we reproduced.

- Also would have been cool to see an animation of the control system in action somehow.

- **Response**: While this would be interesting, we believe it would be time consuming and is not at the heart of what this course was about.

8.3. Critique by Group 88

Group 29 gave a great introduction on the feedback linearization of nonlinear systems and some basic mathematical methods. They set a goal of learning transformation and linear matrix from data. They used a Deep Learning model of Koopman operator and introduced its loss function. Then they used Tensorflow to achieve their goal. Their codes work well to show the result of their model. Also, they did a lot of work on mathematical analysis. Overall, it is a good presentation. But, it will be better to make some improvement.

Some improvement/unclear

- What is the reason to choose Koopman operator? What are the advantages and disadvantages of this models while comparing to others?

- **Response**: Koopman Operator Theory is the most studied approach for the problem we considered.

- They could explain more detailed on the result plots they got. And we think some of the results are not very good (like the Trajectory Prediction). What is the reason? How to improve the result.

- **Response**: We actually think the trajectory prediction results were great.

- The set of training data may be a little bit small. Could they use a bigger one to analyze the model?

- **Response**: For the Koopman problem, we increased the data size to 13,000. For feedback linearization, we kept the data-size at 5000 which achieved sufficiently good results, and is comparable to the data-size used in [6]