Transfer Learning and Dynamical Systems Estimation

Jing Liu
Research Report, Summer 2020
Supervisor: Prof. Hsiu-Chin Lin
McGill University School of Computer Science
September 23, 2020

Abstract

The dynamical system (DS) $\dot{x} = f(x)$ can define the motion of a robot in a given space. If the DS is stable, it can ensure that the robot reaches a target destination no matter where it begins its trajectory. Currently, the Linear Parameter Varying (LPV-DS) method proposed by N. Figueroa and A. Billard robustly learns globally asymptotically stable (GAS) DSs from nonlinear input trajectories [1]. The goals of this project, which builds on LPV-DS, are twofold. First, we sought to explore transfer learning, the learning of a dataset using parameters from a different dataset. An application could be a robot hand learning to grasp a ball, and using that knowledge to grasp a jar. Second, we wanted to experiment with different regression methods and compare them to LPV-DS. The transfer learning experiments did not prove successful with our particular datasets, but using deep learning (DL) for DSs showed promise. While the resulting DS was not GAS, the errors were overall lower than those for LPV-DS (number of runs $n = 20$). This result shows that DL, while not without limitations, is a technique that should be explored further in the context of DSs.

1 Introduction and Background

A major topic in robotics is controlling motion. For example, we may want to ensure a robotic arm reaches a target destination, regardless of its starting point. Mathematically, this motion could be described with an **autonomous dynamical system** [2]. **Dynamical systems (DSs)** are systems that change according to predefined variables and rules [3]. In this case, the variables could be position and time, and the rules could include convergence, the constraint that the arm must reach its target. The DS could then return a velocity for every input position and time, forming a vector field of motion with a specified target, as $\dot{x} = f(x, t)$. Removing the time dependency creates an **autonomous** system of the form

$$\dot{x} = f(x), \tag{1}$$

an example of which is visualized in Figure 1. Such systems are more robust because time instructions can be delayed [2].

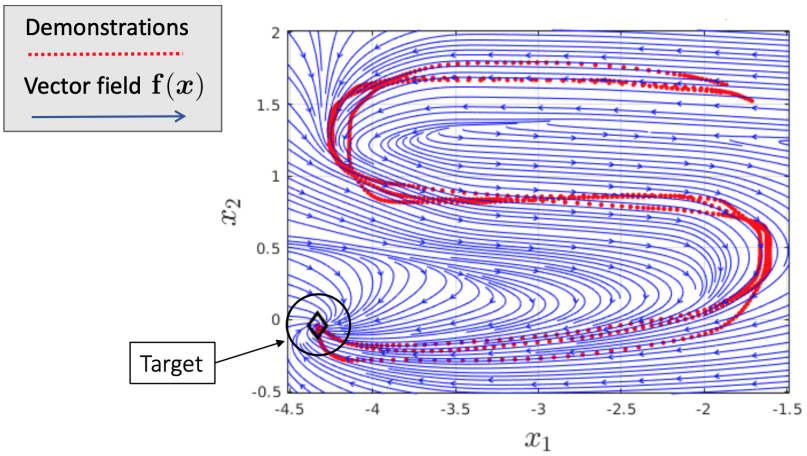

Figure 1: An example DS, with velocity vectors given for every point in space [4].

My work is built on [1] [1], which addresses the following problem: given an input dataset of trajectories pointing towards a target, we want to characterize the underlying DS over the entire space while maintaining convergence to the target. In other words, we have a regression problem where we must learn $f(x)$.

This notion of convergence can be more strongly defined through global asymptotic stability. In general, **equilibrium** refers to any state in a system that does not change [5]. A **stable** equilibrium is one that is not affected by small disturbances; the system will always return to equilibrium [5]. For the DS $\dot{x} = f(x)$, a point x^* is an **equilibrium point** if $f(x^*) = 0$ [6]. A DS is **globally asymptotically stable (GAS)** if every trajectory in the space ends at x^* [6].

Global asymptotic stability can be defined through **Lyapunov theory**, a technique that allows us to draw conclusions about a DS's trajectories without explicitly knowing them [6]. Suppose $V(x) : \mathbb{R}^N \to \mathbb{R}$ is a continuously differentiable function such that $V(x) \to \infty$ as $||x|| \to \infty$, satisfying the following conditions:

1. $V(x^*) = 0$,
2. $V(x) > 0$ for all $x \neq x^* \in \mathbb{R}^N$,
3. $\dot{V}(x^*) = 0$,
4. $\dot{V}(x) < 0$ for all $x \neq x^* \in \mathbb{R}^N$.

Then $V(x)$ is called a **Lyapunov function**, and we can say the DS is GAS at $x^* \in \mathbb{R}^N$ [4] [2].

Conditions 1 – 3 are easy to enforce if $V(x)$ is taken to be the distance function $V(x) = \frac{1}{2}(x - x^*)^T(x - x^*)$, since this quantity must always be ≥ 0. A **linear DS**, the simplest case, can be written as $\dot{x} = Ax + b$ [4]. To ensure $\dot{V}(x) < 0$ for all x (except x^*), we can enforce two conditions.

[1] Source code: https://nbfigueroa.github.io/pc-gmm-ds-learning/
[2] Proof in [6].

First, $A^T + A \prec 0$ [7]. In other words, $A^T + A$ must be **negative definite**, meaning it is a $d \times d$ real symmetric matrix such that $x^T(A^T + A)x < 0$ for all vectors $x \in \mathbb{R}^d$ [8]. If A is a non-symmetric matrix, A is negative definite if and only if its symmetric part $A_S = \frac{1}{2}(A+A^T)$ is negative definite [7]. Thus, the condition $A^T + A \prec 0$ allows for $x^T Ax < 0$, without A having to be symmetric [7].

Second, $b = -Ax^*$ [7]. With those in place, we have:

$$\begin{aligned}
\dot{V}(x) &= \frac{dV}{dt} = \frac{dV}{dx}\frac{dx}{dt} \\
&= \frac{1}{2}\frac{d}{dx}\left[(x - x^*)^T(x - x^*)\right]\dot{x} \\
&= (x - x^*)^T\dot{x} \\
&= (x - x^*)^T(Ax + b) \\
&= (x - x^*)^T[A(x - x^*) + Ax^* + b] \\
&= (x - x^*)^T A(x - x^*) \\
&< 0 \text{ for all } x \neq x^*.
\end{aligned}$$

The above is a simplified version of the proof found in [7], which addresses the following, more general type of DS. A **nonlinear DS** can be represented as a combination of linear DSs, i.e.

$$\dot{x} = f(x) = \sum_{k=1}^{K} \gamma_k(x)(A_k x + b_k), \tag{2}$$

where $\gamma_k(x)$ is a mixing function that determines the contribution of each DS in the sum [4]. The full mathematical reasoning can be found in [7], but each DS is actually a Gaussian distribution; thus, the underlying DS is actually a mixture of K Gaussians. This form of regression is known as a **Gaussian Mixture Model (GMM)**, a probabilistic model most often used for multimodal populations. Intuitively, the data is categorized into clusters, with each point being assigned a likelihood of belonging to each cluster [9].

The clusters' parameters (i.e. mean μ and variance σ^2 in 1-D, or mean vector μ and covariance matrix Σ in higher dimensions) can be learned using **expectation maximization (EM)**, an algorithm that iteratively updates the probabilities of each data point belonging to each cluster and re-estimates parameters. (More background on EM can be found in [9].) The authors of [7] then propose the **Stable Estimator of Dynamical Systems (SEDS)** algorithm, which computes the optimal parameters under the constraint of global asymptotical stability. However, SEDS has two issues, according to [1]: highly non-linear motions are reproduced inaccurately, and the number of Gaussians K must be specified in advance [4]. As such, the authors of [1] submit a new approach that addresses these concerns.

They consider $f(x)$ (Equation 2) as a linear parameter-varying system, meaning it changes according to a function of time-varying parameters [10]. In this approach, dubbed **Linear Parameter Varying (LPV-DS)**, the quadratic Lyapunov function is parameterized as $V(x) = (x - x^*)^T P(x - x^*)$, where P is a symmetric, positive definite matrix [1]. P is introduced to allow for highly nonlinear and non-monotonic trajectories, and the stability condition is modified to $A^T P + PA \prec 0$ [1].

However, this parameterization presents difficulty for optimiziation with SEDS, and so the authors decouple the mixing function parameters $\theta_\gamma = \{\pi_k, \mu^k, \Sigma^k\}_{k=1}^K$ (the mixing weight, mean, and covariance for each Gaussian, respectively) from those of the linear DS, $\theta_f = \{A_k, b_k\}_{k=1}^K$ [4]. The learning algorithm must also be changed, as EM produces clusters with non-adjacent data points; in other words, we want to ensure that points within a cluster follow a linear DS model [1]. The authors resolve this by imposing a physically-consistent similarity metric derived from cosine similarity [1]. The resulting LPV-DS method both automatically learns the proper K, and is able to embed large non-linearities, thus resolving the two main concerns with SEDS [4].

My objectives for this project were to: (1) explore the possibility of **transfer learning**, i.e. storing variables learned from one dataset and using them to learn a similar dataset's DS quicker or more accurately; and, (2) experiment with other regression methods for creating a GAS DS, namely multivariate linear regression and deep learning.

2 Process

2.1 Measuring Performance

In the experiments, performance is measured by three errors (as originally used in [1]):

1. Prediction **root mean squared error (RMSE)**, $RMSE = \frac{1}{M}\sum_{m=1}^{M} ||y_m - \hat{y}_m||$, where y_m are the true values and $\hat{y}_m$ are the predicted values. [1]. The RMSE is simply the normalized sum of squared errors.

2. Prediction **cosine similarity**, $\dot{e} = \frac{1}{M}\sum_{m=1}^{M} \left|1 - \frac{\hat{y}_m^T y_m}{||\hat{y}_m||\,||y_m||}\right|$ [1]. This quantity measures similarity between two vectors by measuring the cosine of the angle between their projections in space [11]. It is only dependent on orientation, not magnitude [11].

3. **Dynamic time warping distance (DTWD)**, a difference measure between two temporal sequences that does not consider time alignment [12].

The first two errors measure the overall difference in shape between the learned DS and the input trajectories, while the DTWD measures the difference between the input trajectories and their generated counterparts [1].

2.2 Procedure

I began by experimenting with transfer learning. Inside `ds-opt`, the main repository from [1], I modified the LPV-DS script to enable transferring parameters between runs. In particular, μ^k and Σ^k are learned during the GMM fitting. These are originally initialized by running k-means++, but my modification allows for them to be initialized with stored values from a previous run. A_k and b_k, the parameters of the linear DS learned during optimization, are also stored and transferred. The results of this experiment are described in Section 3.1.1.

Next, I picked out a specific area from two datasets. These areas roughly correspond to one of the Gaussians that is learned as the GMM is fitted in both datasets. We hoped

that these smaller, partial datasets would better accommodate transfer learning due to their increased linearity and similarity. The results are found in Section 3.1.2.

Another interesting question is whether parameter transferring could be used to learn a new dataset with fewer datapoints. In this experiment, a dataset's parameters were stored, and then another dataset was learned using those parameters, but with progressively fewer points each run. I tracked errors vs. dataset size to judge if transfer learning could reduce the number of reference datapoints needed to learn a DS. The results are described in Section 3.1.3.

The second goal of this project was to explore with different regression methods and compare their performance to LPV-DS. One of the reasons behind the complexity of LPV-DS is the constraint that the resulting DS must be GAS. I began by removing that constraint and fitting a simple linear regression to the data; because the outputs are 2-D (x- and y-components of velocity), I used multivariate linear regression (MVR). In this case, MVR is the same as solving the system $y = xA$ for the matrix of coefficients A, where both x and y are 2-D. The results of MVR compared to LPV-DS are detailed in Section 3.2.1.

Finally, we wanted to know if using deep learning (DL) could directly return a GAS DS. I used a 5-layer neural network with a test and validation proportion of 0.2. The model is fit over 500 epochs with a mini-batch size of 20, using the Adam optimizer. The parameters were primarily chosen through a tutorial and are thus suitable for further experimentation.

The model uses a regression layer with a custom loss function designed to account for the constraints given by the Lyapunov function. The MSE (RMSE squared) is the original cost function that we want to minimize when learning the DS. We can subject the MSE $||y - \hat{y}||^2$ to a constraint, say $g(x) < 0$, by adding a **barrier function** as such:

$$||y - \hat{y}||^2 - \log(-g(x) + 1). \tag{3}$$

The barrier function is plotted in Figure 2. We can think of $y = \text{Re}(-\log(-g(x) + 1))$ as a penalty that increases as we approach the bounds of the constraint. $g(x) = 0$ would produce a penalty of 0, while $g(x) > 0$ quickly rises to infinity, indicating a clear preference for values of $g(x)$ that are ≤ 0. In general, barrier functions can be used to replace inequality constraints with a penalizing term in the loss function, to allow for easier calculation [13].

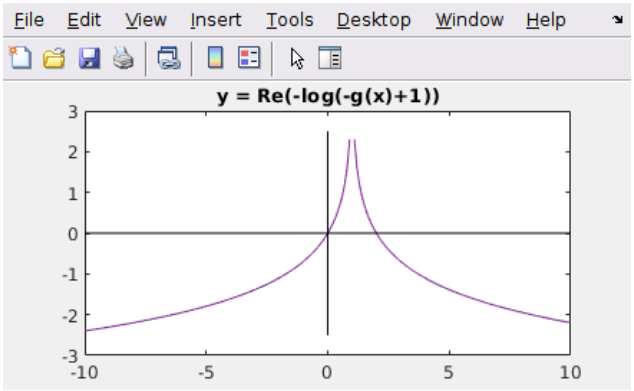

Figure 2: A logarithmic barrier function for the constraint $g(x) < 0$.

We can adapt the above to our situation using the Lyapunov function $V(x)$. Recall

condition 4 for stability: $\dot{V}(x) < 0$ for all $x \neq x^* \in \mathbb{R}^N$, which indicates a preference for moving towards the target at all points in space. Thus, for $V(x) = (x - x^*)^T(x - x^*)$, we have $\dot{V}(x) = 2(x - x^*)^T\dot{x} = 2\dot{x}^T(x - x^*) = 2y^T(x - x^*) \equiv g(x)$. The full loss function is then

$$L(y, \hat{y}) = ||y - \hat{y}||^2 - \log(-2y^T(x - x^*) + 1). \tag{4}$$

Its derivative, $\partial L/\partial \hat{y} = 2(y - \hat{y})$, is used for backpropogation. The results from our DL model with this custom loss function are described in Section 3.2.2.

3 Results and Discussion

3.1 Transfer Learning

3.1.1 Full Datasets

The LASA Handwriting datasets provided in `ds-opt` have significant variation in terms of shape and position. I selected two of the most similar datasets I could find, LASA 4 and 5 (Figures 3 and 4, respectively), for our first experiment. The results of transferring parameters for LASA 4 are found in Table 1, while those for LASA 5 are in Table 2.

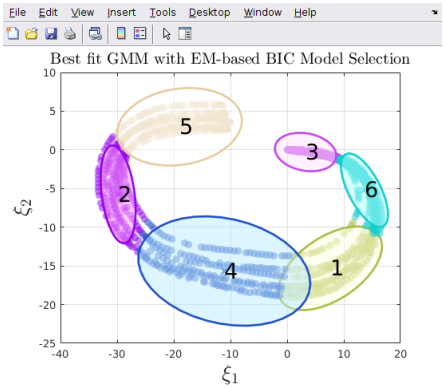

Figure 3: GMM for LASA 4.

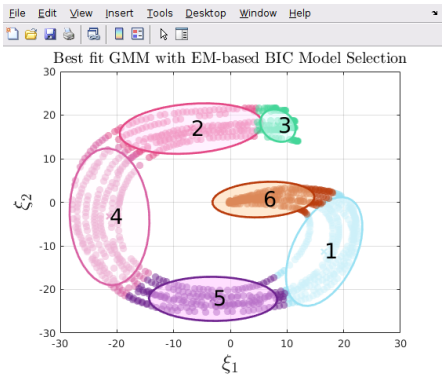

Figure 4: GMM for LASA 5.

	No saved parameters	Parameters transferred from LASA 5
RMSE	3.3277 ± 0.2668	3.4127 ± 0.1642
$\dot{e}$	0.1733 ± 0.0106	0.1619 ± 0.0143
DTWD	2320.4178 ± 354.0189	2183.6123 ± 303.9546
Time	3.8781 ± 1.4654	4.1609 ± 0.3454
K	5.8000 ± 1.4832	6

Table 1: Results of transferring parameters for LASA 4 ($n = 5$).

There are several issues. Firstly, the errors are improved only in certain cases, meaning this approach may not be robust enough. Furthermore, K varies over different runs; parameters, however, can only be transferred between models fitted to the same K. As such, I

	No saved parameters	Parameters transferred from LASA 4
RMSE	4.3069 ± 0.3253	4.4508 ± 0.2711
$\dot{e}$	0.0277 ± 0.0077	0.0292 ± 0.0082
DTWD	2736.1824 ± 464.3733	2650.4514 ± 311.8088
Time	3.7889 ± 0.8093	3.9178 ± 0.4769
K	5.8000 ± 0.8367	6

Table 2: Results of transferring parameters for LASA 5 ($n = 5$).

rounded the source dataset K to the nearest integer and fixed that to be K when running on the target dataset. Finally, it should be noted that for this and subsequent LPV-DS experiments, I used the simpler EM algorithm found in [7] for optimization, as opposed to the latest one from [1]. It would be worth exploring the implications of transfer learning for the latter.

3.1.2 Partial Datasets

Since the full datasets were too different for transfer learning, we decided to try with partial datasets. LASA 7 and 8 both have a vertical, linear section approximately bounded by $x = [22, 30]$ and $y = [-10, 20]$. These areas roughly correspond to the location of a consistently-learned Gaussian during GMM fitting. Note that the bounds were determined by eye, so their precision could be improved to find two even more similar datasets.

The partial datasets are shown in Figures 5 and 6. The results are shown in Tables 3 and 4, this time for 20 runs each.

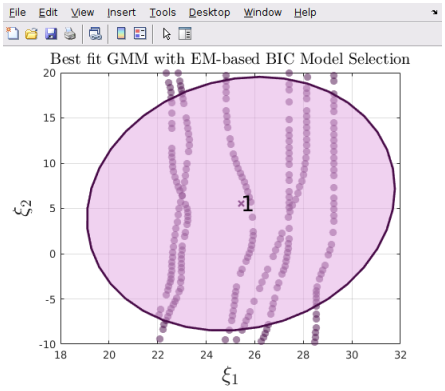

Figure 5: LASA 7 partial dataset.

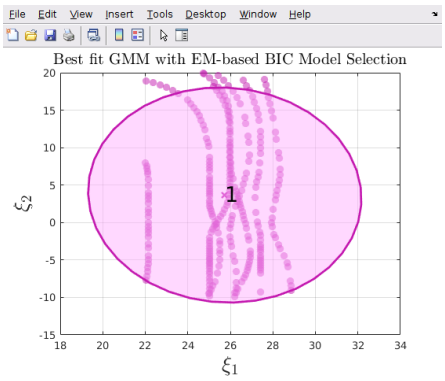

Figure 6: LASA 8 partial dataset.

Unfortunately, there were still no overall decreases in error or computation time. For further experiments, datasets with greater similarity could be used–though already, it's apparent that this method of transfer learning is not particularly robust.

3.1.3 Learning with Less Data

For this experiment, we wanted to see if parameters learned from a previous dataset would facilitate learning a new dataset with fewer points. For learning without previous parameters,

	No saved parameters	Parameters transferred from Dataset 8
RMSE	3.6158 ± 0.2137	3.6064 ± 0.2714
$\dot{e}$	0.0091 ± 0.0015	0.0090 ± 0.0012
DTWD	31204.1015 ± 1272.5490	31325.1116 ± 1073.8942
Time	0.6038 ± 0.0590	0.6278 ± 0.0573
K	1	1

Table 3: Results of transferring parameters for LASA 7 partial ($n = 20$).

	No saved parameters	Parameters transferred from Dataset 7
RMSE	4.7190 ± 0.2186	4.7562 ± 0.2739
$\dot{e}$	0.0142 ± 0.0019	0.0146 ± 0.0025
DTWD	31323.5994 ± 1869.3137	31453.0627 ± 1809.1113
Time	0.6623 ± 0.0868	0.6736 ± 0.1544
K	1	1

Table 4: Results of transferring parameters for LASA 8 partial ($n = 20$).

we expect errors to decrease as dataset size increases, because more sample points should give a more representative picture of the underlying DS. The goal of this experiment was to confirm this hypothesis for transfer learning. Note that a test set was fixed, i.e. each run used the same test points, to reduce random error.

The results from LASA 7 are seen in Figure 7. Judging from a basic linear regression, the orange lines, corresponding to learning with saved parameters, are steeper (cosine similarity and DTWD) or produce lower errors overall (RMSE). The first observation suggests that the impact of dataset size is greater in transfer learning than in regular LPV-DS. The second observation points to transfer learning being successful for lowering RMSE, for this particular dataset.

Meanwhile, the results from LASA 8 (Figure 8) are more mixed. While the blue lines (no saved parameters) are similar to those in LASA 7, the orange lines all have positive slopes, indicating that a larger number of data points does not improve the learned DS. I'm not sure what is responsible for this seemingly contradictory result, because even if LASA 7's parameters are not a good starting point for learning LASA 8, the error should not increase with more data.

Of course, this is a simplistic analysis–other considerations might include the goodness of fit of the regression lines to the data, other types of regression, and larger dataset sizes–though that would require new datasets entirely.

3.2 Other Regression Methods

3.2.1 Multivariate Linear Regression

MVR is the most straightforward form of regression, yet it still produces reasonable-looking results in our case. For example, compare the DSs shown in Figures 9 and 10: both are

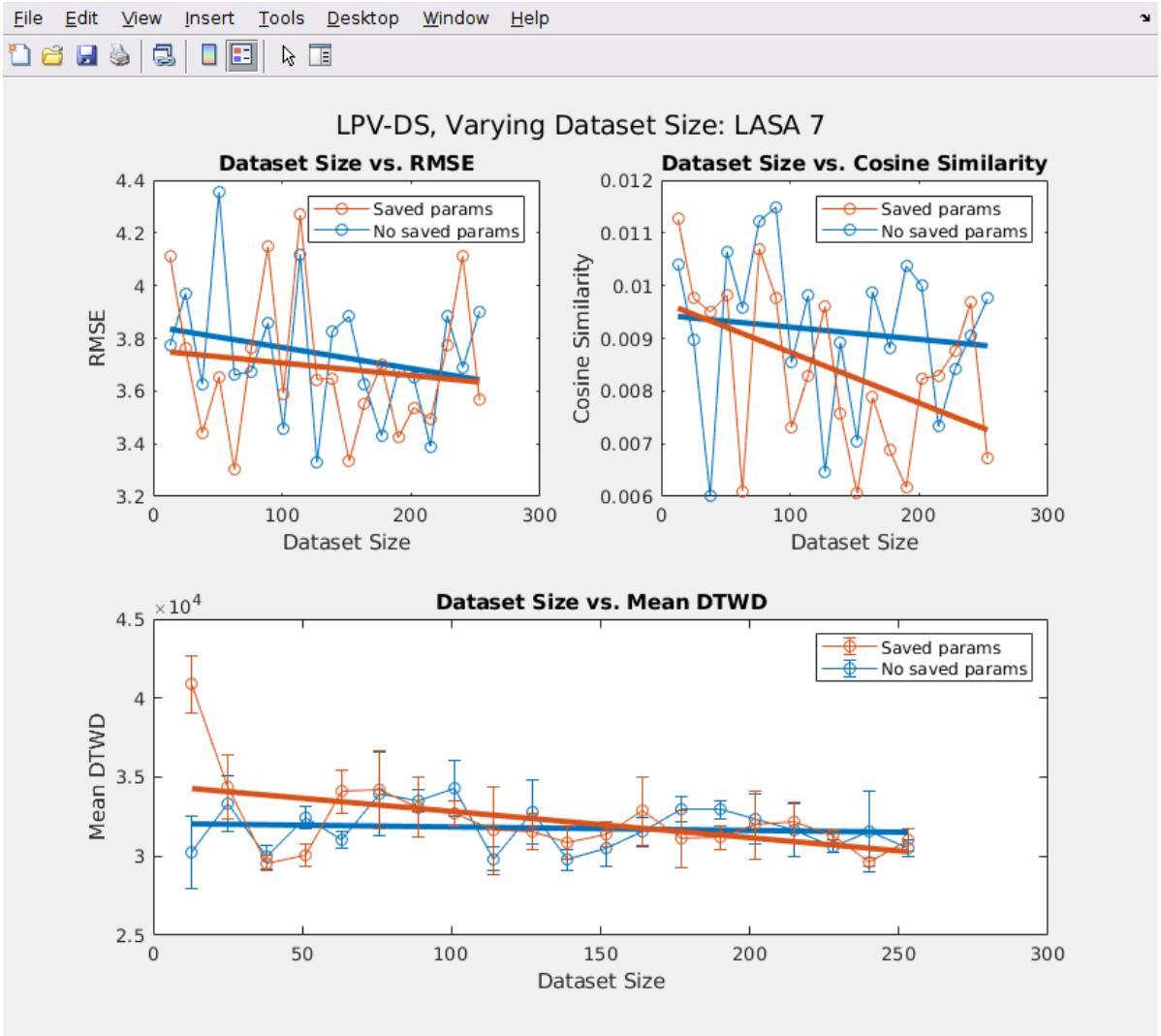

Figure 7: Errors as a function of dataset size for LASA 7, learned without saved parameters as well as using parameters from LASA 8.

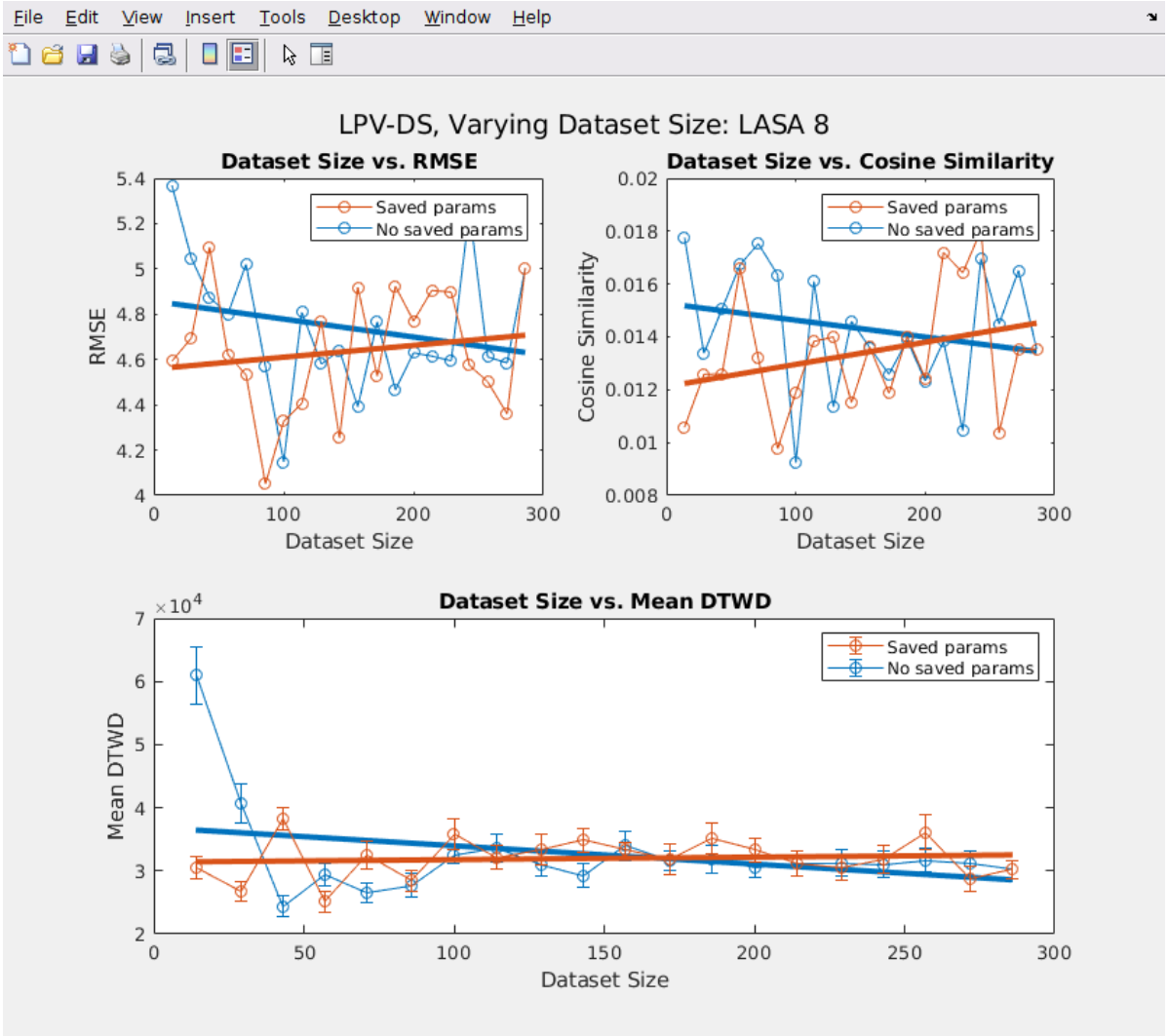

Figure 8: Errors as a function of dataset size for LASA 8, learned without saved parameters as well as using parameters from LASA 7.

GAS. The issue however is that MVR cannot account for large nonlinearities, and so the resultant DS will differ significantly from the input trajectories. Accordingly, the errors are much higher (see Tables 5 and 6). Overall, MVR is too simplistic for these datasets, though it might be suitable for more linear ones. Additionally, the constraint of global asymptotic stability is not explicitly defined in this method, so it may not be preserved in all cases.

	LPV-DS	MVR
RMSE	6.5518 ± 0.0380	9.7766 ± 0.2111
$\dot{e}$	0.0332 ± 0.0003	0.1676 ± 0.0150

Table 5: LASA 7, LPV-DS vs. MVR ($n = 20$).

	LPV-DS	MVR
RMSE	5.7361 ± 0.0397	9.4553 ± 0.2562
$\dot{e}$	0.0605 ± 0.0022	0.2192 ± 0.0149

Table 6: LASA 8, LPV-DS vs. MVR ($n = 20$).

3.2.2 Deep Learning

From Tables 7 and 8, we might conclude that our DL model produces better results than LPV-DS since the errors are noticeably lower, with the exception of cosine similarity for LASA 7. However, Figure 11 suggests a different interpretation. While the output DS may be closer to the input trajectories than with LPV-DS, the resulting DS is not GAS. In particular, the areas where there is no input data suffer from broken or enclosed trajectories, meaning the custom regression layer was unable to enforce stability. This result shows that LPV-DS is still better for learning a GAS DS; while its errors may be higher, it successfully accounts for the entire space regardless of where its input trajectories lie. That being said, DL may nevertheless be useful due to its ability to closely model input data; different dataset types and methods of ensuring stability should be further explored.

	LPV-DS	DL
RMSE	6.5518 ± 0.0380	4.3056 ± 0.1963
$\dot{e}$	0.0332 ± 0.0003	0.0558 ± 0.0165

Table 7: LASA 7, LPV-DS vs. DL ($n = 20$).

	LPV-DS	DL
RMSE	5.7361 ± 0.0397	4.3577 ± 0.2900
$\dot{e}$	0.0605 ± 0.0022	0.0403 ± 0.0115

Table 8: LASA 8, LPV-DS vs. DL ($n = 20$).

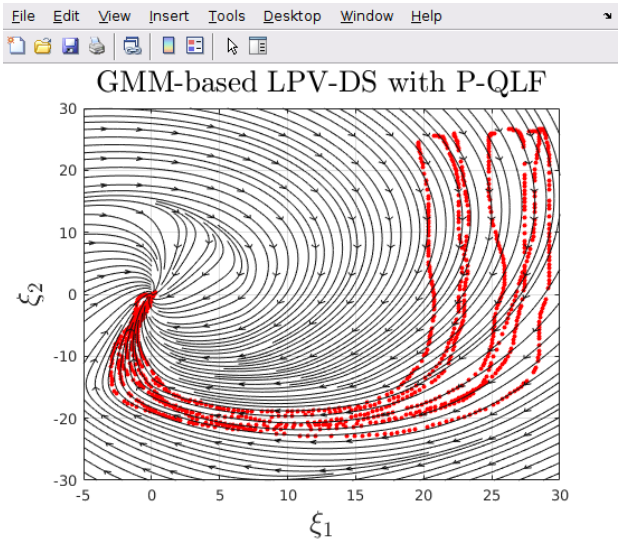

Figure 9: LASA 7 learned with LPV-DS.

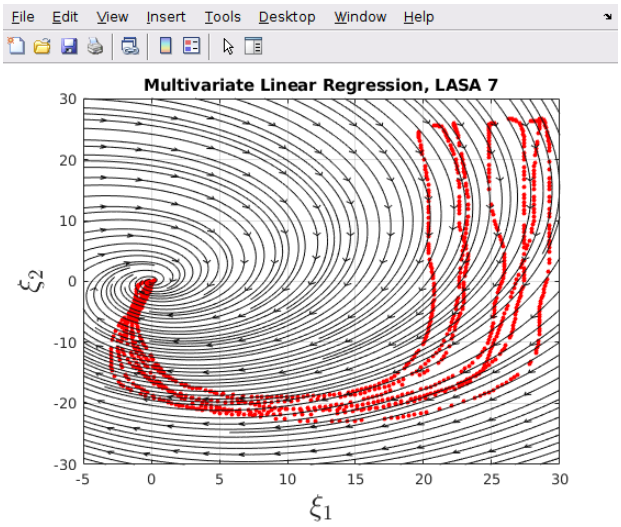

Figure 10: LASA 7 learned with MVR.

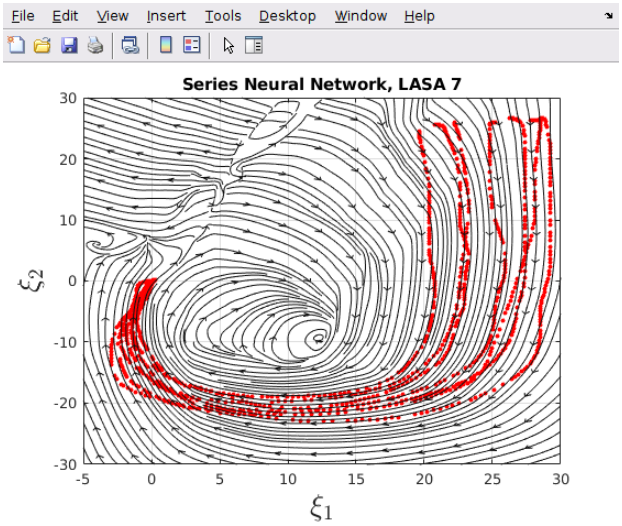

Figure 11: LASA 7 learned with DL.

4 Conclusion

In summary, two extensions of LPV-DS were undertaken in this project. First, transfer learning was unsuccessful for both large nonlinear datasets and small linear ones, though it remains to be seen what would happen with the newer optimization algorithm proposed in [1]. Learning with less data gave contradictory results, and should be explored further. Second, MVR gave us a glimpse at perhaps the simplest way of learning a DS, though it did not return lower errors than LPV-DS. DL, on the other hand, is able to reduce errors overall but fails the GAS constraint due to discontinuities in areas without input data. While LPV-DS is superior to DL for now, the latter could be a logical next step if the constraints are defined in a different way.

References

[1] N. Figueroa and A. Billard, "A physically-consistent bayesian non-parametric mixture model for dynamical system learning," in *Proceedings of The 2nd Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, A. Billard, A. Dragan, J. Peters, and J. Morimoto, Eds., vol. 87. PMLR, 29–31 Oct 2018, pp. 927–946. [Online]. Available: http://proceedings.mlr.press/v87/figueroa18a.html 1, 2, 3, 4, 7, 12

[2] S. S. M. Salehian. [Online]. Available: https://epfl-lasa.github.io/TutorialICRA2019.io/documentation/Introduction.html 1

[3] Libretexts, "3.1: What are dynamical systems?" Aug 2020. [Online]. Available: https://math.libretexts.org/Bookshelves/Applied_Mathematics/Book:_Introduction_to_the_Modeling_and_Analysis_of_Complex_Systems_(Sayama)/03:_Basics_of_Dynamical_Systems/3.01:_What_are_Dynamical_Systems? 1

[4] S. S. M. Salehian. [Online]. Available: https://epfl-lasa.github.io/TutorialICRA2019.io/documentation/Learning.html 2, 3, 4

[5] A. Sharov, Dec 1996. [Online]. Available: https://web.ma.utexas.edu/users/davis/375/popecol/lec9/equilib.html 2

[6] S. Boyd. [Online]. Available: https://stanford.edu/class/ee363/ 2

[7] S. M. Khansari-Zadeh and A. Billard, "Learning stable nonlinear dynamical systems with gaussian mixture models," *IEEE Transactions on Robotics*, vol. 27, no. 5, pp. 943–957, 2011. 3, 7

[8] "Appendix c: Positive semidefinite and positive definite matrices," Mar 2007. [Online]. Available: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470173862.app3 3

[9] E. M. Liu. [Online]. Available: http://ethen8181.github.io/machine-learning/clustering/GMM/GMM.html 3

[10] "Lpv system." [Online]. Available: https://www.mathworks.com/help/control/ug/linear-parameter-varying-models.html 3

[11] S. Prabhakaran, "Cosine similarity - understanding the math and how it works? (with python)," Apr 2020. [Online]. Available: https://www.machinelearningplus.com/nlp/cosine-similarity/ 4

[12] H. Ravichandar, I. Salehi, and A. Dani, "Learning partially contracting dynamical systems from demonstrations," ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 369–378. [Online]. Available: http://proceedings.mlr.press/v78/ravichandar17a.html 4

[13] M. S. Gockenbach. [Online]. Available: https://pages.mtu.edu/~msgocken/ma5630spring2003/lectures/bar/bar/node2 5

A Code

The code for this project can be found at https://github.com/musicbyjing/learning_dynamical_systems.