CAPTURING DYNAMICAL SYSTEMS USING DEEP LEARNING AND UNDERSTANDING OPTIMAL DATA SIZE IN TRAINING TIME SERIES PREDICTION MODELS

A Thesis
Submitted to the Faculty
in partial fulfillment of the requirements for the
degree of

Bachelor of Arts

in

Mathematics

by Brian Wang

Advised by Professor Yoonsang Lee
Dartmouth College
Hanover, NH

May 2023

Abstract

This thesis examines capturing dynamical systems (specifically, the Lorenz 96 model) using deep learning. Specifically, this work looks into situations where only partial observations of a dynamical system are available, and predictions need to be made on the future states of these partial observations using past data. The ultimate goal of this work is to build a deep learning model that can simulate a dynamical system under various levels of sparsity in observations and use this model to study the nature of the relationship between sparsity and optimal trajectory length of historical data to use for the model. We leverage existing and create new qualitative and quantitative analysis techniques that allow us to determine the optimal trajectory length for each level of sparsity and compare results across different modes of experiments. We find that the relationship between sparsity and optimal trajectory length follows a non-linear relationship, with there being an exponential decaying relationship between the number of dimensions in our observations and the optimal trajectory length for our deep learning model (for $dt = 0.01$). Further study is needed to understand how this pattern between sparsity and optimal trajectory length generalizes to simulating dynamical systems of varying complexity using varying time steps with varying types of sparsity for our observations.

Acknowledgements

This work would not have been possible without the guidance and support of a number of people. First and foremost, I would like to thank Professor Yoonsang Lee. After taking his Probability and Statistical Inference Course in the Winter of 2022, I became interested in his research. He has been an incredible advisor, both inside and outside of research, over the last two years, giving me valuable advice in my academic, professional, and skiing pursuits. I would also like to thank Jinman Park, who has helped me tremendously in guiding me through my research in Professor Lee's Lab. I am forever grateful for the mentorship these two have provided and this thesis would not have been possible without them.

I would also like to thank the entire Math Department and Computer Science Department for the opportunity to study and work with brilliant professors over the past few years, allowing me to dive deep into subject areas that I'm passionate about.

Last but not least, I would also like to thank my friends (particularly, Ryan, Genghe, Andrew, and Jessica for being the best study buddies I could ask for while completing this thesis) and my family (Dad, Mom, and David) for their continued support throughout my time at Dartmouth. I couldn't have done this without all of your love and support over the last few years.

Contents

Chapter 1

Motivation

Section 1.1

Motivation

We live in a world where we have to make predictions based on incomplete observations. For instance, we make predictions on the weather and the price of stocks without knowing all the variables that influence these items of interest, the values for those variables, and the function that relates those variables to our items of interest. For instance, the price of a stock VOO could be governed by some formula with 1 trillion variables: $VOO = 0.9*$VOO-PREV-DAY - $0.00754*$BPS-RAISE $+ ... + 0.0000000312*$NYC-ANALYST-A-SLEEP-HOURS.

However, it is not possible for us to always know these variables. Sometimes, we need to rely solely on the values of our items of interest to make predictions (e.g. making predictions for VOO based on the last few values of VOO).

Within the index fund VOO, there are 500 stocks. These 500 stocks are all related to each other in some way (whether by being in the same industry, being affected by the same macroeconomics headwinds/tailwinds, etc.). If we are trying to predict the prices of all 500 stocks, we may use the previous 5 hours of past prices to predict the

next hour's prices. If we are trying to predict the prices of 200 stocks, and only have information on 200 stocks, we may have to use more previous hours of past prices since we are now missing some information we could have derived from seeing the other 300 stocks. But how much further back in time do we need to look to make accurate predictions?

Section 1.2

Problem Statement

The goal of this thesis is to create guidelines for how much historical data to look at when predicting values for a partially observed system. We aim to use the sparsity of the data as the main guideline for determining the amount of historical data (i.e. trajectory length) to look at. We also aim to determine whether the level of sparsity of data and the optimal trajectory length follows a non-linear or linear relationship.

Section 1.3

Problem Framing

In this thesis, our complex system will be the Lorenz 96 dynamical system. Our partial observations of this system will consist of observing every other point, observing every three points, etc. We will have data on this system for every 0.01 seconds ($dt = 0.01$) and for each level of sparsity, we will test various trajectory lengths to find the optimal length of historical data to use. For our predictions, we aim to not only predict the next few partial observations correctly, but to have our prediction system capture the overall dynamics of the Lorenz 96 model, being able to simulate the values of the variables and the interactions between different variables in the system correctly over time.

Chapter 2

Related Work

Previous research has already studied how prediction models can learn complex dynamical systems [12]. They consider the performance of these prediction models in settings where there aren't full observations (i.e. when we are working with partial observations). Considering the performance of these models in the face of partial observations is crucial, as in the real world we rarely have access to a full system (and even if we do, it could be computationally expensive to consider all variables of a system). These studies have also looked into new methods to qualitatively and quantitatively assess the performance of these prediction models, which we will adopt when building our own prediction models [12].

However, previous studies do not consider another facet of data missingness beyond sparsity of variables: the length of historical data passed into models. This is particularly important to consider, as it is often tricky to know how much historical data to use when making predictions on time series data. Having guidelines on trajectories to try for certain levels of sparsity would significantly help expedite the model tuning process.

Our approach looks into the performance of prediction models in settings with only partial observations of systems, as we examine the optimal length of historical

data and its relationship with the sparsity of the data at hand, recognizing that a lack of data could lead to low quality predictions (because the model is not able to understand the relationship between the variables) while too much data could also lead to low quality predictions (because the model struggles to separate out what values are important to consider and struggles to pick out an overall trend).

Chapter 3

Methods

Section 3.1

Experiments

3.1.1. Sparsity (s)

We aim to build models that will be able to use values of various subsets of variables in the Lorenz 96 model and predict the future values for all variables in that subset. We run experiments for for sparsity ranging from $s = 1$ (full observations) to $s = 8$ (observing every 8 variables. For a given s, we will observe the variables x_0, x_s, x_{2s}, ..., observing a total of $\lceil \frac{N}{s} \rceil$ variables.

For instance, for sparsity $s = 2$, we will consider the values for 20 ($\lceil \frac{N}{s} \rceil$) variables (i.e. x_0, x_2, ..., x_{38}) and try to predict future values for those 20 variables.

3.1.2. Trajectory Length (t_l)

For a time $t + 1$, our deep learning model will use the values of our variables in the t_l previous time steps (i.e. $x_{t-(t_l-1)}, x_{t-(t_l-2)}, ..., x_t$) to predict x_{t+1}. We aim to find some mathematical relation between between sparsity and the optimal length for historical data. We aim to see if this relationship tends to be linear or non-linear.

Section 3.2

Data

3.2.1. Lorenz 96 Model

Formulated by Edward Lorenz [9], the Lorenz 96 model is a dynamical system used in data assimilation problems. With N variables (with $N \geq 4$) and a forcing constant F, the Lorenz 96 model is governed by the following differential equations:

$$\forall i \in \{1, 2, ...N\}, \frac{dx_i}{dt} = (x_{i+1} - x_{i-2})x_{i-1} - x_i + F \tag{3.1}$$

For the edge cases, the Lorenz 96 model defines $x - 1 = x_{N-1}$, $x_0 = x_N$, and $x_{N+1} = x_1$

In our experiments, we use $N = 40$ and $F = 8$ (a value for F known to cause chaotic behavior). A sample of the first three variables of the Lorenz 96 is shown in Figure 3.1.

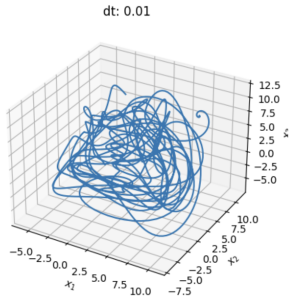


Figure 3.1: Sample 3-dimensional Phase Plot of Lorenz with dt=0.01

3.2.2. Simulating Lorenz 96 and Training Data Generation

Using an initial condition of (8.0081, 8, 8, ..., 8), we simulate the Lorenz 96 using the Python scipy library's *solve_ivp* function (a high-order numerical solver) using the 'RK45' integration method [3]. In the Runge-Kutta-Fehlberg method, steps are taken

using the fifth-order accurate formula (using local extrapolation), working under the assumption that the error is controlled due to accuracy of the fourth-order method [8]. This integration method is suitable for our complex dynamical system.

Using the initial condition and the *solve_ivp* function, we integrate the Lorenz 96 function from t = 0s to t = 10000s with t = 0.01 (so 1,000,000 time steps).

As mentioned earlier, for a time $t+1$, our deep learning model will use the values of our variables in the t_l previous time steps to predict x_{t+1}. During the training of our model, the model will be used to predict the values of our variables for 5 steps. For time steps $t+2$ to $t+5$, the model will use a combination of the actual values of our variables (values for x before time $t+1$) and predicted values of our variables (values for x after time t) as the values for the previous t_l time steps. For instance, to predict the values of the variables at time t+2, the model will use $x_{t-(t_l-2)}, x_{t-(t_l-3)}, ..., x_t$ and $\hat{x}_{t+1}$ as the value of the variables in the previous t_l time steps (the motivation for predicting 5 steps instead of just 1 is explained later in the section on the loss function).

As such, from our our data of 1,000,000 time steps, we randomly select 10,000 sequences of size $t_l + 5$).

3.2.3. Testing Data Generation

Prediction is carried out to $T = 500s$ (50,000 time steps) from a new initial condition (8.0081, 8, 8, ..., 8). We give our model the values for each of the variables of interest for the first t_l time steps (to have data from $x_1, x_2, ..., x_{t_l}$). The model predicts the values of the variables of interest one time step at a time, using its past predictions as the input (e.g. to predict the value of the variables at time step $t_l + 2$, the model will use $x_2, x_3, ...,$ and x_{t_l} and $\hat{x}_{t_l+1}$.

Section 3.3

Predictive Model

3.3.1. Model Architecture

We build a custom deep learning model using tensorflow [6] that uses three hidden layers with 200 neurons each (1 LSTM layer and 2 Dense layers) and one output Dense layer whose number of nodes is equal to the dimensionality of our data. We arrive at this architecture after significant testing via the full model, arriving at the specific number of neurons after determining that it is a good medium between underfitting and overfitting. In this case, the LSTM layer is used because of its ability to learn long-term dependencies between time steps in time series data [2]. The LSTM layer is especially appropriate in the case of trying to capture the dynamics of the Lorenz 96 model, as the derivatives of each variable are governed by values of variables in previous time steps. Each layer uses the "tanh" activation function, which is the default used for recurrent neural networks [11] (other activation functions, like "relu" are better suited for computer vision problems).

Figure 3.2 shows an example of the model summary for the model used when working with data of sparsity 6 (where we are working with 7 variables).

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 lstm_15 (LSTM)               (None, 200)               166400

 dense_45 (Dense)             (None, 200)               40200

 dense_46 (Dense)             (None, 200)               40200

 dense_47 (Dense)             (None, 7)                 1407

=================================================================
Total params: 248,207
Trainable params: 248,207
Non-trainable params: 0
```

Figure 3.2: Sample model architecture for s=6

3.3.2. Loss Function

In deep learning, a loss function is used to compare the predicted output values of a model to the target values (the actual output values from the training data).

To prevent model drift, a recurrent loss function is implemented to measure the performance of the model of predicting the values for the variables of interest in time steps $t+1$ to $t+5$ (different from the usual loss function that only looks at predictions for time step $t+1$). Using the method explained in section 3.2.2, predictions $x_{t+1}, x_{t+2}, ..., x_{t+5}$ are made for time steps t+1 to t+5. Afterwards, the following function is applied to calculate loss:

$$loss = \sum_{t_i=t+1}^{t+5} \sum_{i=1}^{N} (x_{i,t_i} - \hat{x}_{i,t_i})^2 \tag{3.2}$$

Our training loss is calculated as the sum of the losses among the 5,000 training samples. Our validation loss is calculated as the sum of the losses among the 5,000 testing samples.

Without a recurrent loss function, the model's predictions may "spiral out of control" as it compounds its mistakes when using past predicted values to predict future values, as seen in Figure 3.3.

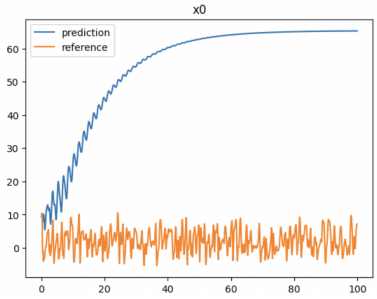


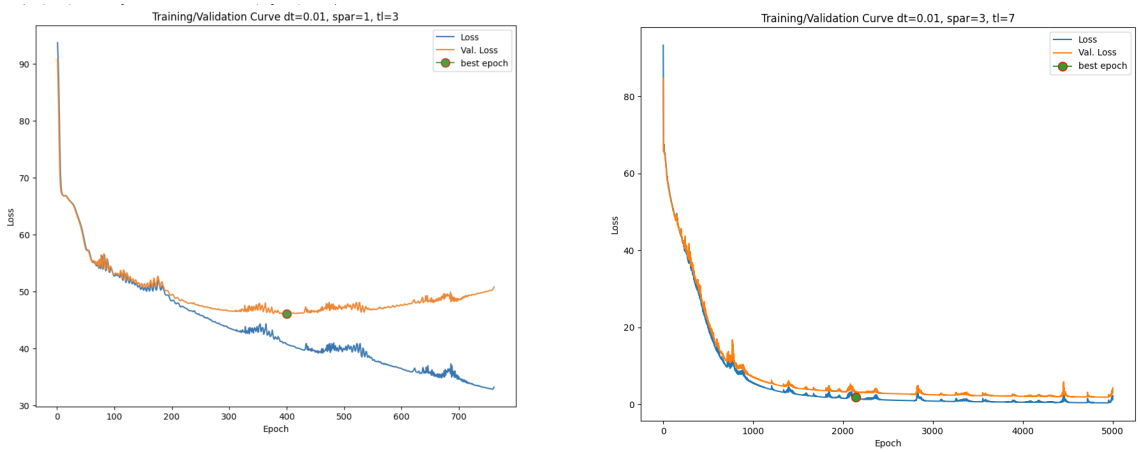
Figure 3.3: Predicted vs. actual values for x_0 (with model trained on non-recurrent loss function)

3.3.3. Model Training

The network hyperparameters are optimized using the Stochastic Optimization Method Adam, which is well-suited for contexts like our work due to the size of the data we are using [7]. We utilize a constant learning rate of 10-3 for 2,000 epochs.

A train-test-split of 0.5-0.5 is used, with half of the 10,000 samples created during the training data generation used for training the model and the other half used to validate the training of the model. We use the resulting model of the epoch that has the lowest validation loss. Figure 3.4 shows the training and validation loss curves for the model training using s = 1 (full observations) and s = 3 (partial observations) data. For the training of both of these models, we use the resulting model of the epoch label with a green dot since this represents the epoch where the validation curve is the lowest. In Figure 3.4(a), we are preventing over fitting by taking the model from the epoch before the training and validation curve diverge.

To save on precious computing resources, we have safeguards in place to prevent the model for training for too many epochs. If our model's validation loss increases by 10% from the minimum validation loss at least 100 epochs away from the minimum point (to account for temporary spikes), we will halt the training of the model before 5000 epochs (as seen in Figure 3.4(a)). To account for variation when the validation loss is lower, we prevent an unnecessary early cutoff by scaling our threshold for cutting off training to be 1.5 times the minimum validation loss for minimum validation losses that are less than 10 and 2 times the minimum validation loss for minimum validation losses that are less than 5.



(a) (b)

Figure 3.4: Training/validation loss curves from: **(a)** $s = 1, t_l = 3$ **(b)** $s = 3, t_l = 7$

Section 3.4

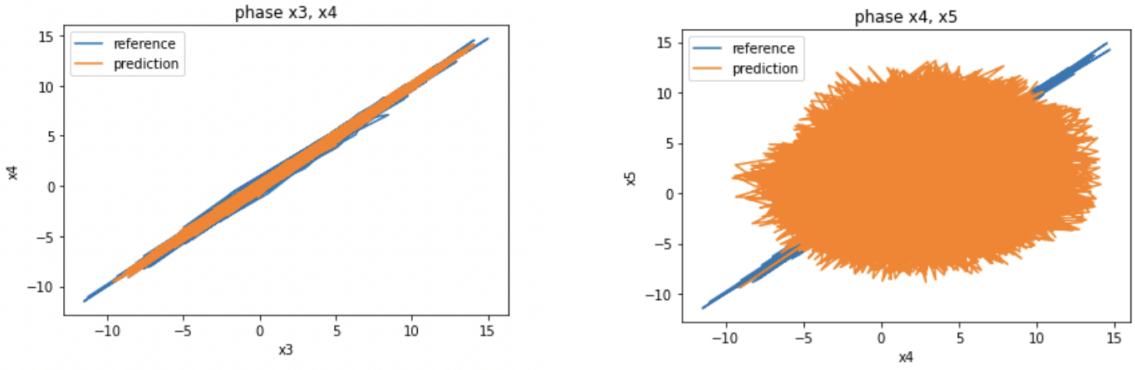
Qualitative Evaluation

3.4.1. Phase Plot

We use two-dimensional phase plots to understand how the model is simulating the relationships between different variables. These phase plots track the trajectory of the predicted and actual values of 2 variables at a time.

These phase plots are valuable because they not only tell us whether the model is properly capturing the range of values for each variable, but also tells us whether these values are appropriate given the value of other related variables. For instance, in Figure 3.5, the range of values for the variables are all captured properly, with both variables ranging between -10 to 10 in each figure. However, our phase plot in Figure 3.5(b) is able to show us that at times, the model is not capturing the relationship between two variables properly (as it is in Figure 3.5(a)), with times where the values of the related variables are quite different from one another (though they should be

close to the $x_i = x_j$ line).

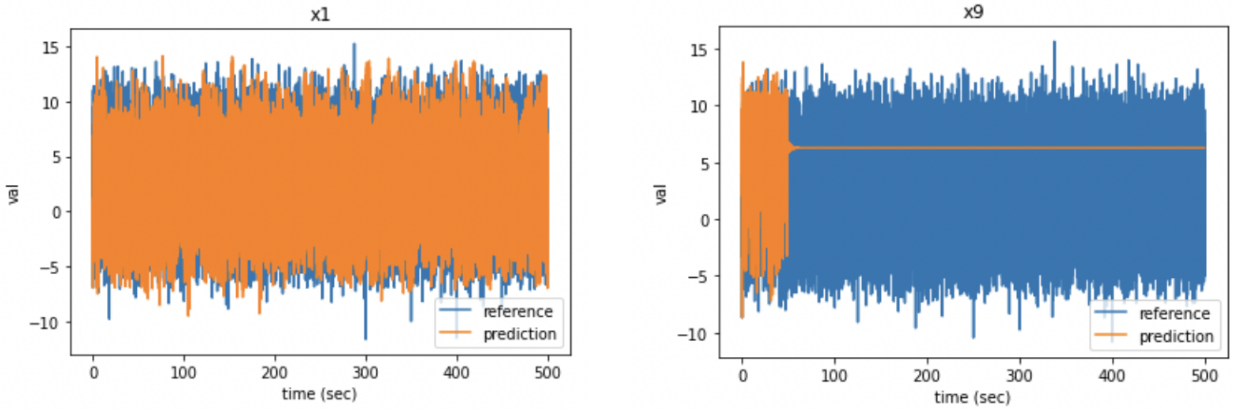


(a) (b)

Figure 3.5: Sample phase plots

3.4.2. Time Series Plot (Full)

In addition to looking at the interaction between variables, we also look at the trajectories of each variable's predicted and actual values over time. We consider time series plots to see whether our model is able to capture the oscillations and range of values for each variable (like we see in Figure 3.6(a)). These time series plots can also tell us when our model gets "stuck" predicting the same values like seen in Figure 3.6(b).



(a) (b)

Figure 3.6: Sample time series plots ($t = 0$ to $t = 500$)

3.4.3. Time Series Plot (Partial)

Because the full time series plots contain 50,000 time steps, it is hard to see how our model is performing on a more micro-level scale. As such, we consider the time series plots from $t = 1$ to $t = 2$ (avoiding the first 100 time steps because our model uses the actual values for the first t_l time steps before it starts making its own predictions). As shown in Figure 3.7, these granular time series plots can help us better understand how are predictive model is simulating the frequency and amplitude of the oscillations of each of our variables.

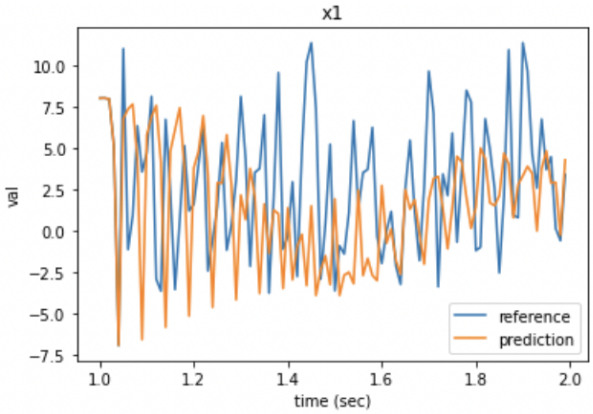
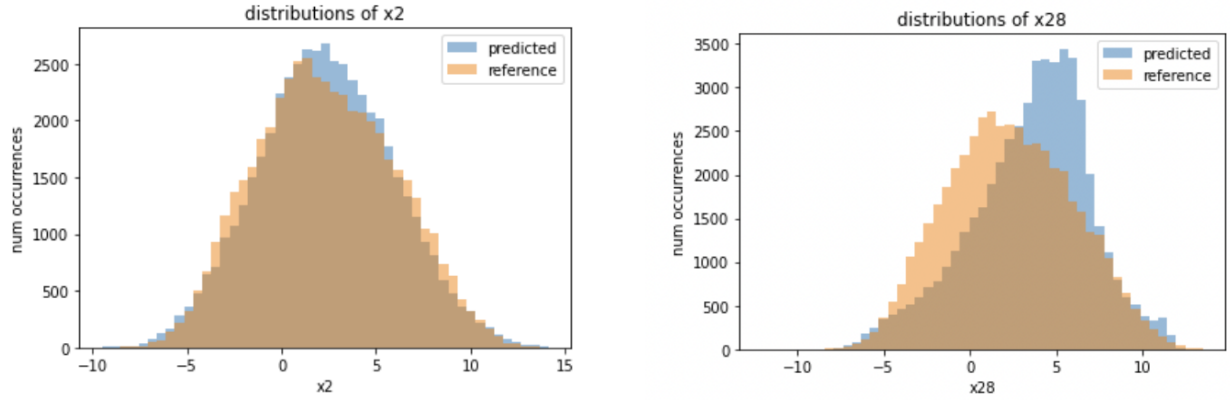


Figure 3.7: Sample time series plots ($t = 1$ to $t = 2$)

3.4.4. Histogram of Values

Meanwhile, in addition to assessing whether the predictive models are capturing the range of values properly using previous qualitative analysis techniques, histograms can help us understand whether the deep learning models are predicting values of each variable with the right frequency (i.e. following the actual distribution of the values of the test trajectory). For instance, though both histograms in Figure 3.8 show that the model is predicting the right range of values, our histogram is able to show that in Figure 3.8(b), the distribution of predicted values has an unnecessary left skew. Meanwhile, Figure 3.8(a) demonstrates the model capturing properly both the range of values and the relative frequency of values in that range.

For the histogram analysis for each variable, we divide the range of values into 50 bins and plot the predicted and reference distribution of values on top of each other.



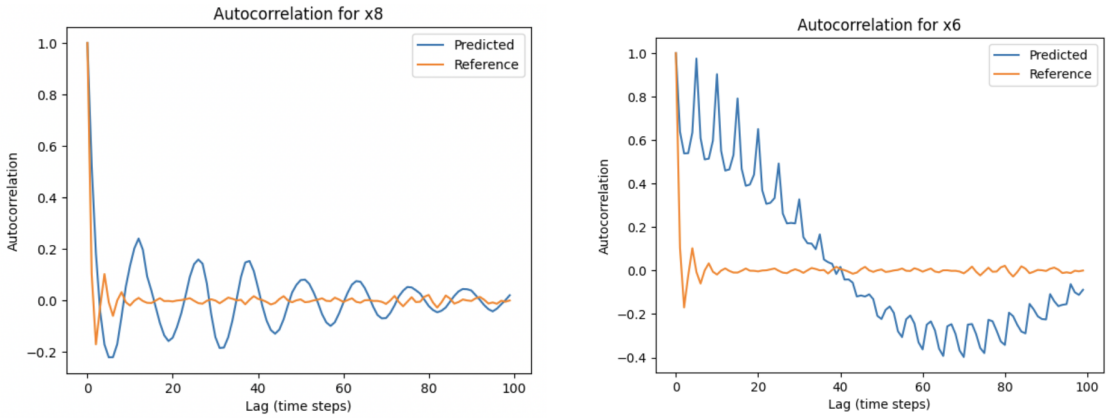
(a) (b)

Figure 3.8: Sample histograms

3.4.5. Autocorrelation Plot

Autocorrelation provides a measurement for the correlation between a time series plot and its lagged counterpart. We compute the autocorrelation for each variable using the statsmodels acf function [4], computing the autocorrelation for 0 to 100 lags (representing a lag of $t = 0$ to $t = 1$) for both the predicted and actual test trajectories.

If the plots of the autocorrelation values are similar, we can conclude that the model is able to capture the relationship between a variable and its past values. For instance, though not perfect, the autocorrelation of the predicted values in Figure 3.9(a) follow a similar trend to that of the actual values, taking an initial dip and oscillating near 0 as we approach time $t = 100$. On the other hand, Figure 3.9(b) shows that the predictive model is poor at relating previous values to new values, as the autocorrelation is positive for up to 40 time lags (when it should be near 0 after about 5 lags) and becomes negative for time lags 40 to 100.



(a) (b)

Figure 3.9: Sample autocorrelation plots

Section 3.5

Quantitative Evaluation

3.5.1. Point-wise error

We look at the reference test trajectories and compare that against the trajectories predicted by the model. We normalize by the number of testing points we have and by the number of dimensions we are working with (which varies depending on sparcity) so that we have comparable metrics to work with.

$$point - wise - error = \frac{\sum_{t_i=1}^{50000} \sum_{i=1}^{N} x_{i,t_i} - \hat{x}_{i,t_i}}{50000 * N} \tag{3.3}$$

This may not be the best measurement, since a model that perfectly predicts the trajectory (but is one time step behind or ahead) would be severely penalized by this metrics. Yet, this could still tell us if in general, the model is predicting roughly the correct range of values and if the predicted trajectories are following the true trajectories somewhat well.

3.5.2. Relative Entropy (Kullback-Leibler divergence)

Earlier, the qualitative analysis on the histogram of values helped us understand whether the distribution of predicted values generally aligned with the true distribution of values in the test trajectory. We want a more rigorous understanding of how well the predicted and actual distribution of values line up.

We use the Kullback-Leibler divergence, which with $numBins = 50$, actual distribution p, and predicted distribution q is computed as

$$D = \sum_{b=1}^{b=50} p_b * log(\frac{p_b}{q_b}) \tag{3.4}$$

That is, entropy is a measure of the likelihood that we could arrive at a distribution q after sampling from a distribution p. The lower the number, the more likely it is (i.e. less entropy means that the two distrbution p and q are more similar).

To compute K-L Divergence, we use the scipy library's *entropy* function [5]. To deal with potential zeroes for p_b and q_b, we set $p_b, = 0.00001$ when $p_b = 0$ to prevent an undefined value for log (since log(0) is undefined) and $q_b = 0.00001$ when $q_b = 0$ to prevent a divide by zero error.

Then, to make our measure of K-L Divergence comparable among experiments with different sparsity levels (so different numbers of dimensions), we normalize the sum of the K-L divergences by the number of dimensions there are.

$$entropy = \frac{\sum_{i=1}^{N} D_i}{N} \tag{3.5}$$

3.5.3. Histogram Differences

Because of the nature of the Kullback-Leibler Divergence formula, there is a wide range of values and the computed entropy is prone to being heavily affected by outliers (as the entropy for one variable could significantly affect the average entropy). Thus,

we also use histograms to get a "pseudo-measure" of entropy that is on a scale of 0 to 1.

To compute our "histogram differences" metric for each variable, we first find the minimum among all predicted and actual values and the maximum among all predicted and actual values. From there, we separate the area between the minimum and maximum into 50 equally-sized bins. For each bin b, let's say there are p_b occurrences of values in the predicted distribution and a_b occurrences of values in the actual distribution. We compute the histogram differences using the following equation

$$HD = \frac{\sum_{b=1}^{b=50} |p_b - a_b|}{50000 * 2} \tag{3.6}$$

Thus, two completely overlapping distributions will have a histogram difference score of 0 and two completely separate distributions will have a histogram difference score of 1.

Among all of our variables for a given sparsity, we look at the average, minimum, and maximum HD to understand how well in general and at the extremes that our predicted model is capturing the distribution of values of the test trajectory.

3.5.4. Approximate Entropy

We already have measurements that indicate how well our model is predicting the range of values. However, we also need to consider how well the model simulates the fluctuations of the time series data. We use Approximate Entropy to quantify the amount of regularity and the unpredictability of fluctuations over time-series data. Smaller values indicate that the data is more regular and predictable, while higher values indicate the opposite. The Approximate Entropy is calculated by a 6-step algorithm [10]. We use a default embedding dimension of 2 and the default Chebyshev distance metric.

We use the EntroPy [1] implementation of the approximate entropy function to compute the approximate entropy for the predicted (aep) and actual (aea) values for each of our variables. We calculate the average approximate entropy error by performing the following calculation

$$approx - entropy - error = \frac{\sum_{i=1}^{N} \frac{|aep_i - aea_i|}{aea_i}}{N} \tag{3.7}$$

Chapter 4

Results and Discussion

Section 4.1

Model with Complete Observations ($s = 1$)

4.1.1. Analysis and Optimal Trajectory Length

For $s = 1$ (full observations), we aim to pick the optimal trajectory length. We run experiments from $t_l = 3$ to $t_l = 15$ and we first perform quantitative analysis to determine what trajectory lengths we should qualitatively analyze.

In Table 4.1, for each metric, we highlight the value that indicates the best predictive model in green and the second best predictive model in orange. From Table 4.1, it is clear that trajectory length 5 is by far the most optimal, leading in 5 of the 6 metrics.

We consider the aforementioned promising $t_l = 5$ and analyze the performance qualitatively. Figure 4.1(a) suggests that our model is doing quite well, capturing both the oscillation and range of values properly. This property is also reflected by the relatively low approximate entropy error, suggesting that our model is good at capturing the unpredictability of fluctuations in the time-series data. At the same time, the more details time plot in Figure 4.2(b) suggests that our model is doing a

Table 4.1: Experiment Results for $s = 1$ with $t_l = 3$ to $t_l = 15$

trajectory length	pointwise error	entropy	average histogram difference	minimum histogram difference	maximum histogram difference	approximate entropy error
3	3.0101	7.6751	0.8994	0.8406	0.9502	0.9132
4	3.3263	2.0117	0.5677	0.5075	0.6373	0.177
5	4.0952	0.0071	0.0379	0.0203	0.0633	0.0551
6	4.2146	7.1802	0.962	0.9502	0.9696	0.9981
7	5.4116	3.4508	0.6165	0.3209	0.9862	0.8827
8	4.2438	6.99	0.9606	0.9442	0.9896	0.9998
9	6.1262	7.1326	0.9732	0.9408	0.9988	0.9991
10	3.9356	0.6126	0.3213	0.1436	0.5447	0.1342
11	5.6605	3.1674	0.9007	0.8069	0.9563	0.947
12	3.9945	5.1344	0.7654	0.5484	0.8798	0.1729
13	5.0146	6.9434	0.9659	0.9419	0.9977	0.9998
14	4.2583	1.0493	0.4133	0.261	0.5714	0.1002
15	5.0195	6.5941	0.9669	0.9412	0.9887	0.9999

somewhat good job of capturing both the amplitude and frequency of the oscillations, though it is far from perfect. The histogram in Figure 4.3(c) suggests that the model is doing a good job capturing the range of values for a variable, and the relative frequency of values inside that range. This fact is also captured by the low entropy and low histogram differences for $t_l = 5$. Finally, the phase plot in Figure 4.3(d) suggests that the model is doing a good job of capturing the interaction between different variables in the model.

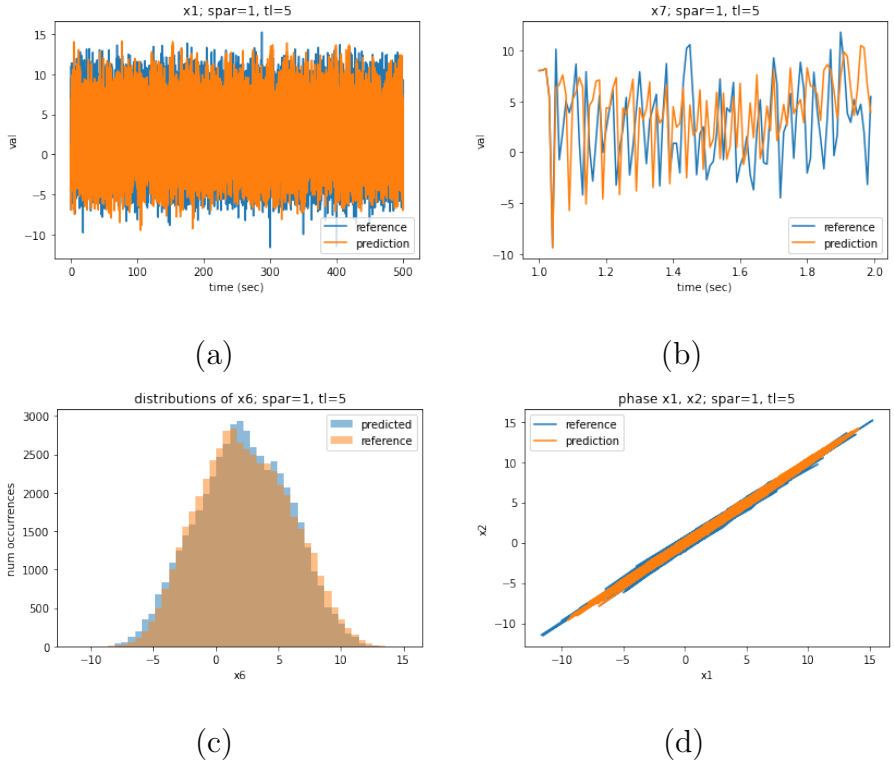
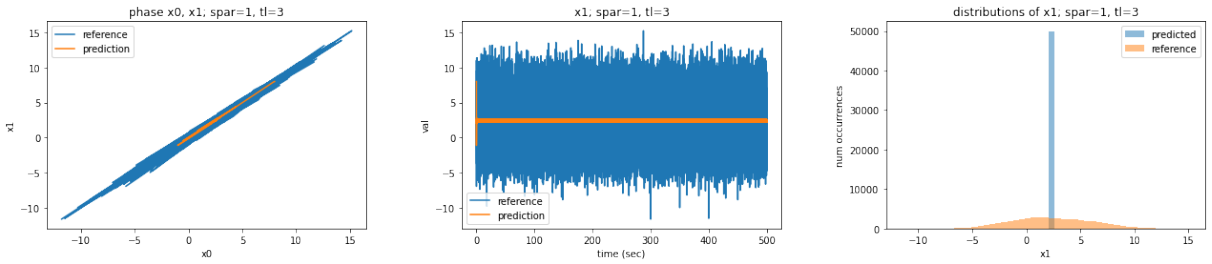


Figure 4.1: Visual Analysis of $s = 1, t_l = 5$ **(a)** Time plot of $t = 0$ to $t = 500$ **(b)** Time plot of $t = 1$ to $t = 2$ **(c)** Histogram **(d)** Phase plot

We then consider $t_l = 3$, which is the only trajectory length to have a metric better than that of $t_l = 5$ for $s = 1$, with the lowest point-wise error. However, looking at Figure 4.2, it becomes clear that $t_l = 3$ is significantly inferior to $t_l = 5$. The model using $t_l = 3$ simply flat-lines (as seen in Figure 4.2(b)), and its status as having the lowest point-wise error is simply because its predictions always lay in the middle of the possible range of actual values. While the model is still able to capture a tiny bit of the nature of the interactions between different variables (seen in Figure 4.2(a)), its predicted distribution of values is significantly off from the actual distribution (seen in Figure 4.2(c)).



(a) (b) (c)

Figure 4.2: Visual Analysis of $s = 1, t_l = 3$ **(a)** Phase Plot **(b)** Time plot of $t = 0$ to $t = 500$ **(c)** Histogram

We then consider $t_l = 10$, which holds the second best metric in 4 categories. Looking at Figure 4.3, the performance of the model using $t_l = 10$ is relatively good, with the phase plot in Figure 4.3(a) showing that the model can capture the interactions between adjecent variables well and Figure 4.3(b) showing that the model is properly capturing the relationship between previous time steps and future time steps with an autocorrelation plot for predicted values that closely traces the autocorrelation plot for the actual values. However, the model's downsides leads us to not consider $t_l = 10$ as the best, as it can be seen that the predicted range fails to capture the upper part of the range of actual values (as seen in Figure 4.3(c)). This can be seen further in Figure 4.3(d), with the predicted values having a positive skew when the distribution of actual values is relatively normal.

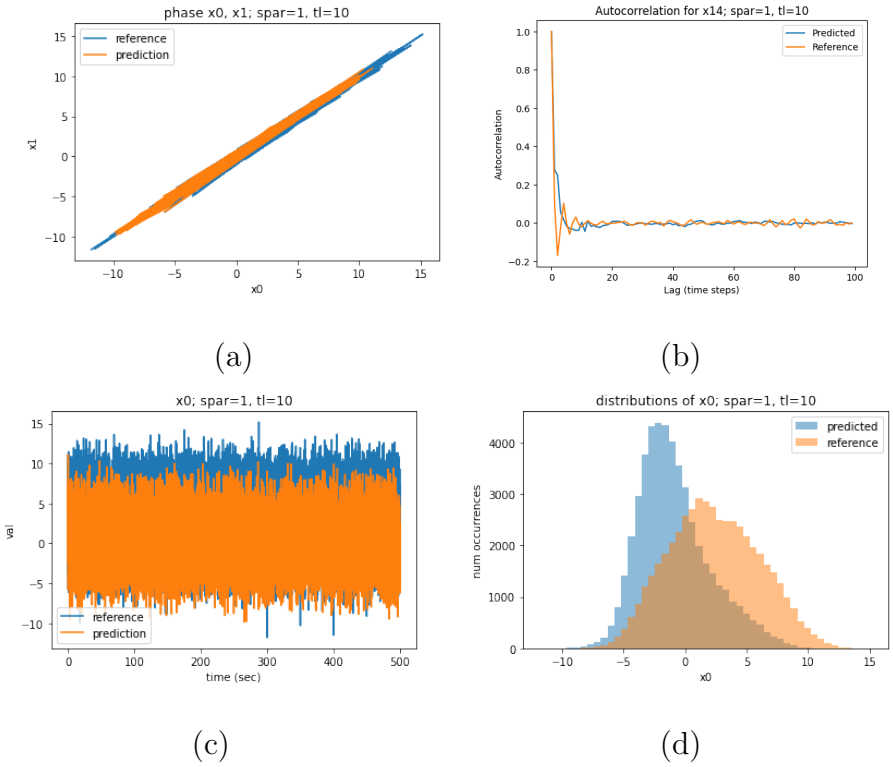


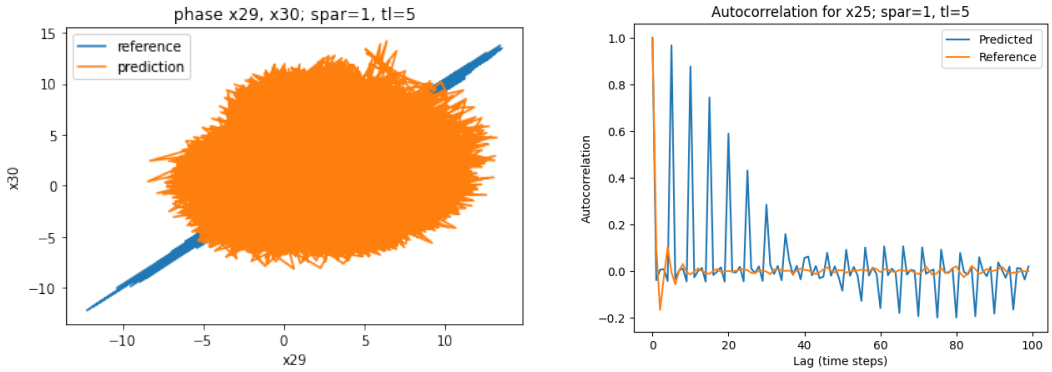
Figure 4.3: Visual Analysis of $s = 1, t_l = 10$ **(a)** Phase Plot **(b)** Autocorrelation Plot **(c)** Time plot of $t = 0$ to $t = 500$ **(d)** Histogram

After this qualitative analysis is performed, we proceed with $t_l = 5$ as our optimal trajectory length for $s = 1$.

4.1.2. Limitations

Although our model performs quite well for $t_l = 5$, there are still some limitations for its performance. 7 of the 39 phase plots appear like the ones shown in Figure 4.4(a), failing to capture the relationship between 2 adjacent variables at times. Moreover, the autocorrelation plot in Figure 4.4(b) shows that the model is not capturing the relationship between past time steps and future time steps that well, as the autocorrelation plot oscillates and flattens way too late (after a lag of about 40 time steps instead of about 5 time steps). However, due to the aforementioned merits of this

model, we proceed with it for the rest of our analysis. The future work section describes potential improvements so that our models do not have these limitations in the future. In an ideal world, our baseline model should be close to perfect when using full observations in the optimal trajectory length.



(a) (b)

Figure 4.4: Limitations of model for $s = 1, t_l = 5$ **(a)** Phase Plot **(b)** Autocorrelation Plot

Section 4.2

Relationship between Sparsity and Optimal Trajectory Length

4.2.1. Determining Best Trajectory Lengths

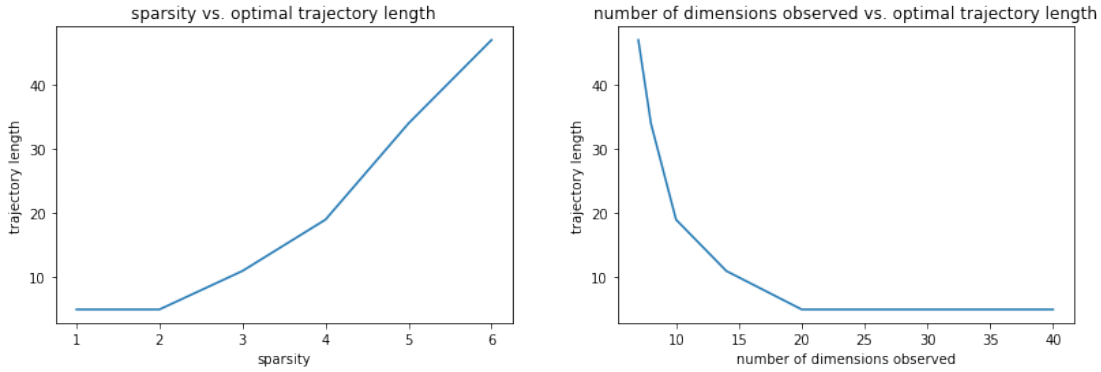
We follow a similar process to the one above to determine the optimal trajectory length for all of our levels of sparsity, trying out many trajectory lengths (up to $t_l = 60$ for $s = 6$), using quantitative analysis to filter out the top few candidate trajectory lengths, and then using qualitative analysis to finalize our selection of the optimal trajectory length for a given sparsity. Table 4.2 describes provides a summary of the models' performance using the optimal trajectory lengths for each sparsity.

When we plot the level of sparsity against optimal trajectory length, we can see

Table 4.2: Experiment Results among optimal $t_l's$ for each of $s = 1$ to $s = 6$

sparsity	trajectory length	pointwise error	entropy	average histogram difference	minimum histogram difference	maximum histogram difference	approximate entropy error
1	5	4.0952	0.0071	0.0379	0.0203	0.0633	0.0551
2	5	4.3646	0.0380	0.0864	0.0287	0.1489	0.0970
3	11	4.3477	0.0277	0.0706	0.0428	0.1207	0.1768
4	19	4.5298	0.1237	0.1791	0.0770	0.2704	0.3603
5	34	4.9963	0.1257	0.1745	0.1150	0.2876	0.2541
6	47	4.3806	0.1451	0.2260	0.1872	0.2459	0.1347

in Figure 4.5(a) that there is an increasing **exponential relationship** between the level of sparsity and the optimal trajectory length. This relationship is made even more apparent when we plot the number of dimensions, which is equal to $\lceil \frac{40}{s} \rceil$ in our case, against the optimal trajectory length in Figure 4.5(b), showing an exponentially decaying relationship between the number of dimensions in our observations to the optimal trajectory length.

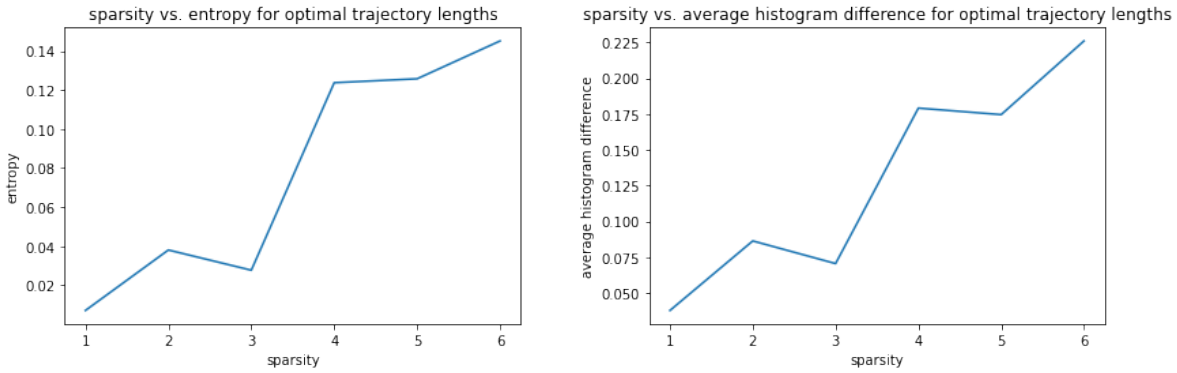


(a) (b)

Figure 4.5: **(a)** Sparsity vs. Optimal Trajectory Length **(b)** Number of Dimensions Observed vs. Optimal Trajectory Length

4.2.2. Diminishing Predictive Accuracy

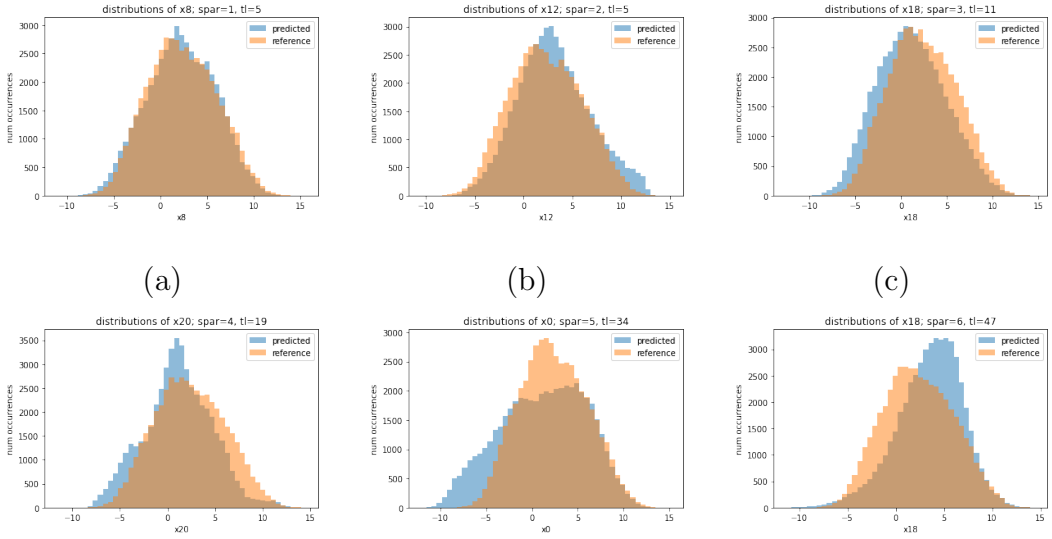
The quantitative analysis suggests that over time, the predictive accuracy for the optimal trajectory length decreases as we increase sparsity. As seen in the Figures 4.6(a) & Figure 4.6(b), entropy and histogram differences increase over time as sparsity increases.



(a) (b)

Figure 4.6: Metric analysis for optimal trajectory length of varying sparsities **(a)** Sparsity vs. Entropy **(b)** Sparsity vs. Average Histogram Difference

Furthermore, the qualitative analysis also suggests that predictive accuracy for the optimal trajectory length decreases as we increase sparsity. As can be seen in the histograms in Figure 4.7 (a)-(f), the predicted distributions veer further and further from the actual distribution of values as sparsity increases, as the predicted values experience some forms of non-normal behavior with skew & inaccurate ranges.



(a) (b) (c)

(d) (e) (f)

Figure 4.7: Sample histograms for optimal trajectory length for **(a)** $s = 1$ **(b)** $s = 2$ **(c)** $s = 3$ **(d)** $s = 4$ **(e)** $s = 5$ **(f)** $s = 6$

Though we were able to find relatively high performing trajectory lengths for $s = 1$ to $s = 6$, we were unable to find an optimal trajectory length for $s = 7$ (searching from $t_l = 45$ to $t_l = 80$), with the best average histogram difference being 0.433 (with a sample from this t_l being shown in Figure 4.8). At this point, we halt the experiments due to the low prediction quality.

However, despite the diminishing predictive accuracy for the optimal trajectory length of a specific sparsity as sparsity increases, it should still be noted that the performance of the model when using the optimal trajectory length is still relatively good when the sparsity is high. For instance, for $s = 5$ (with optimal $t_l = 34$), the average entropy is 0.1257 and for $s = 6$ (with optimal $t_l = 47$), the average entropy is 0.1451. Both of these entropy values are significantly better than the entropy of the predictions made when we had full observations (in $s = 1$) when we did not use the optimal $t_l = 5$, as the second best entropy was 0.6126. Meanwhile,

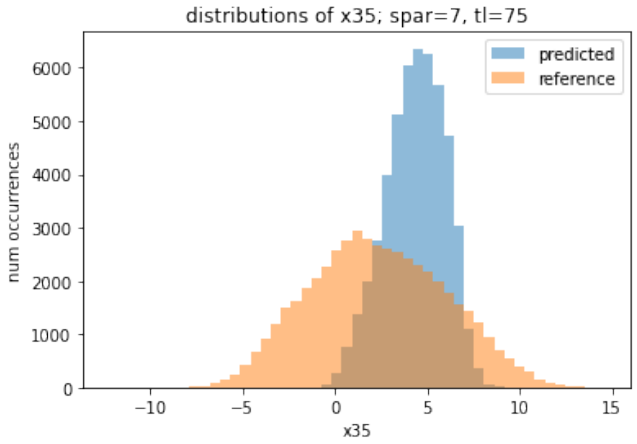


Figure 4.8: Histogram for $s = 7, t_l = 75$

our average histogram differences of 0.1745 (for $s = 5, t_l = 34$) and 0.2260 (for $s = 6, t_l = 47$) are better than the average histogram differences made when we had full observations when we did not use the optimal $t_l = 5$, as the second best average histogram difference was 0.3213. It is worth noting that the *maximum* histogram difference among all variables for $s = 5$ is 0.2876 and for $s = 6$ is 0.2459, which is lower than the second best *average* histogram difference for $s = 1$. Thus, it is reasonable to conclude that the model can still achieve relatively high performance even when our data is quite sparse, observing as low as 7 of the 40 variables in our dynamical system.

Chapter 5

Future Work

There is a variety of work that should be completed to improve the robustness and augment our results.

As seen in the results section, even with full observations, the deep learning model is still far from perfect. To improve the robustness of our testing, we should first improve the baseline model. Though we have already tried various architectures and activation functions, we should consider even more neural network architectures. Furthermore, we could also improve the baseline model by increasing the amount of data that is passed into the model. To do so, we would need to leverage more advanced computing resources (like better GPUs as we are currently using Google Colab Pro).

We should also see whether our pattern for the relationship between sparsity and optimal time length hold for different dt's (other than $dt = 0.01$). This would require another few whole sets of experience, but could prove valuable in validating the relationship between sparsity and optimal time length or providing a different set of rules that govern optimal time length (that are also affected by the value for dt).

We should also consider the relationship between different types of sparsity (with the same level of sparsity) and whether they have different optimal time lengths. For

instance, for $s = 2$, we could have $x_0, x_2, ..., x_{38}$ or we could also have $x_0, x_1, x_2, ..., x_{19}$ and these two different types of sparsity could have drastically different optimal time lengths.

Finally, we should run experiments that consider more complex and less complex dynamical systems (e.g. a less complex dynamical system could be one where only the preceding variable in a vector and the current variable dictate the value of the derivative of the current variable). It would be important to see whether the guidelines for the general pattern between sparsity and optimal time lengths hold in these different systems.

Bibliography

[1] Entropy. https://raphaelvallat.com/entropy/build/html/index.html.

[2] lstmlayer. https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.lstmlayer.html.

[3] Scipy integrate solve ivp. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html#scipy.integrate.solve_ivp.

[4] Scipy stats entropy. https://www.statsmodels.org/dev/generated/statsmodels.tsa.stattools.acf.html.

[5] Scipy stats entropy. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.entropy.html.

[6] Tensorflow. https://www.tensorflow.org/.

[7] Jimmy Ba Diederik P. Kingma. In *Adam: A Method for Stochastic Optimization*. International Conference for Learning Representations, 2015.

[8] Erwin Fehlberg. Eine methode zur fehlerverkleinerung beim runge-kutta-verfahren. Journal of Applied Mathematics and Mechanics, 1958.

[9] Edward N. Lorenz. Predictability - a problem partly solved. Massachusetts Institute of Technology, Cambridge, 2006.

[10] S M Pincus. In *Approximate entropy as a measure of system complexity*. Proceedings of the National Academy of Sciences, 1991.

[11] Siddhartha Mishra Tim De Ryck, Samuel Lanthaler. On the approximation of functions by tanh neural networks. pages 732–750. Neural Networks, 2021.

[12] Victor Churchill & Dongbin Xiu. In *Deep Learning of Chaotic Systems from Partially-Observed Data*, pages 97–119. Journal of Machine Learning for Modeling and Computing, 2016.