Learning and Modeling Chaos Using LSTM Recurrent Neural Networks

Malvern Madondo and Thomas Gibbons
Mathematics, Computer Information Systems Department
The College of St. Scholastica
Duluth, MN 55811
{mmadondo, tgibbons}@css.edu

Abstract

In nonlinear dynamical systems, long-term prediction is extremely challenging. Small perturbations in an initial state can grow exponentially in time and result in large differences in a later advanced state - a behavior known as *chaos.* Chaotic systems tend to have sensitive dependence on initial conditions, much like the Butterfly Effect. Recurrent Neural Networks are dynamic and allow for modeling of chaotic behavior. In this paper, we study and investigate the the modeling and prediction abilities of a Long Short-Term Memory (LSTM) recurrent neural network in dynamical systems with chaotic behavior. In particular, we explore the Lorenz System - which comprises of a nonlinear system of differential equations describing two-dimensional flow of a fluid, and describe an architecture that models the systems' behavior.

Keywords: Chaos, Dynamical Systems, Nonlinear Differential Equations, Long Short-Term Memory RNN, Lorenz System, Recurrent Neural Networks.

1 Introduction

In the early 1960s, MIT meteorologist Edward Lorenz published his paper, *Deterministic Nonperiodic Flow* [1], in which he showed that “a small change in one state of a deterministic nonlinear system can result in large differences in a later state”. This sensitive dependence on initial conditions is known as the *Butterfly Effect* in Chaos Theory. Lorenz described a system of nonlinear differential equations that modeled thermally induced fluid convection in the atmosphere. The significance of the Lorenz System was that relatively simple systems could exhibit complex (chaotic) behavior.

Predicting the future behavior of a dynamic system involves recording and analysing the results of the system over time. These observations, taken at regular time intervals, form what is known as a time series. The estimation of one or several future values of a time series is commonly done using just the information given by the past values of the time series. However, the Lorenz System demonstrates chaotic and random behavior, which makes it extremely hard to predict long-term as small perturbations in the initial conditions are amplified exponentially with time [2, 3]. Mathematical models, such as Artificial Neural Networks (ANN), are often used for making approximate predictions in dynamic systems.

Studies on predicting and modeling chaotic systems using ANN date back to almost a decade ago with the work of Woolley et al [4]. In 2009, the authors carried out a study to determine whether artificial neural networks (ANN) can be used to forecast the outputs of nonlinear dynamic systems. They used the Lorenz System to generate a chaotic data set, on which the ANN with Nonlinear Autoregressive Moving Averages with Exogenous input (NARMAX) was trained. Lyapunov exponents, phase diagrams and statistical analyses were used to evaluate the neural network output and compare it to the actual Lorenz System. Their work largely inspired this paper.

[5] suggests a recurrent NN-based AutoRegressive Moving Average (ARMA) model that performs much better in modeling a Lorenz Chaotic Attractor compared to other equivalent linear and feedforward models. The suggested approach also performs much better in multi-step-ahead predictions. However, this approach is no longer optimal, given it is almost 13 years old, and modern techniques that implement Recurrent Neural Networks and LSTM are more effective.

Recently, [6] investigated time series forecasting by combining a LSTM network and an AutoEncoder (AE) algorithm to capture long-term dependencies across data points and uses features extracted from recent observations to simultaneously augment the LSTM, thereby significantly improving forecasting on chaotic time series. This was proven to be more accurate in predicting both one-step and multi-steps ahead compared to conventional LSTM networks, as explored in this paper.

In this paper, we present an investigation of the modeling and prediction abilities of a traditional Recurrent Neural Network (RNN) and a “Long Short-Term Memory” (LSTM) RNN, when the input signal has a chaotic nature. We study the effectiveness of both networks in modeling the Lorenz System and compare their respective one-step ahead predictions. The overall organization of this paper is as follows: Section 2 describes the Lorenz System; Section 3 describes the network models; Section 4 presents the simulations as well as the training and testing results. Finally, Section 5 presents the conclusions and possible future work.

2 The Lorenz System

While investigating challenges in accurate weather prediction, Lorenz studied the Navier–Stokes equations, which describe the relationship among the pressure, temperature, and density in a moving fluid [3]. Lorenz described a model in which a fluid flows in a container whose top and bottom surfaces are cooled and heated respectively to create a temperature gradient similar to the atmosphere. As the gradient increased, he noted that the fluid transitioned from stationary to steady to chaotic flow. As a result of his study, he came up with the following system of nonlinear differential equations:

$$\frac{dx}{dt} = \sigma(y - x)$$
$$\frac{dy}{dt} = x(\rho - z) - y$$
$$\frac{dz}{dt} = xy - \beta z$$

The variables x, y, and z represent coordinates in 3-dimensional space and are functions of time; that is, x = x(t), y = y(t), and z = z(t).

x - proportional to the intensity of the convection motion

y - proportional to the temperature difference between ascending and descending currents.

z - proportional to the difference of the vertical temperature profile from linearity.

The positive parameters sigma (σ), rho(ρ), and beta (β) are values Lorenz used to exhibit the chaotic behavior of the system when σ = 10.0, ρ = 28.0, β = 8/3. They are respectively proportional to the Prandtl number, Rayleigh (or Reynolds) number, and certain physical dimensions of the layer itself such as the width to height ratio of the container. Although the parameters are usually varied, especially ρ, we use the same values in our experiment and to generate the data, we solve for the Lorenz Equations, augmenting Project Jupyter's *Lorenz System Notebook* [7].

First, we import the computation and display functions from the IPython, NumPy, Matplotlib and SciPy libraries. We then define a function that can integrate the differential equations numerically and then plot the solutions. The function takes in the following arguments: N=10, *max_time*= 4.0, σ = 10.0, ρ = 28.0, β = 8/3 where N are the number of datapoints which we computer over *max_time* time steps. Plotting the results, we obtain the following figure, known as the Lorenz Attractor, which is a set of numerical values toward which a system tends to evolve, for a wide variety of starting conditions of the system.

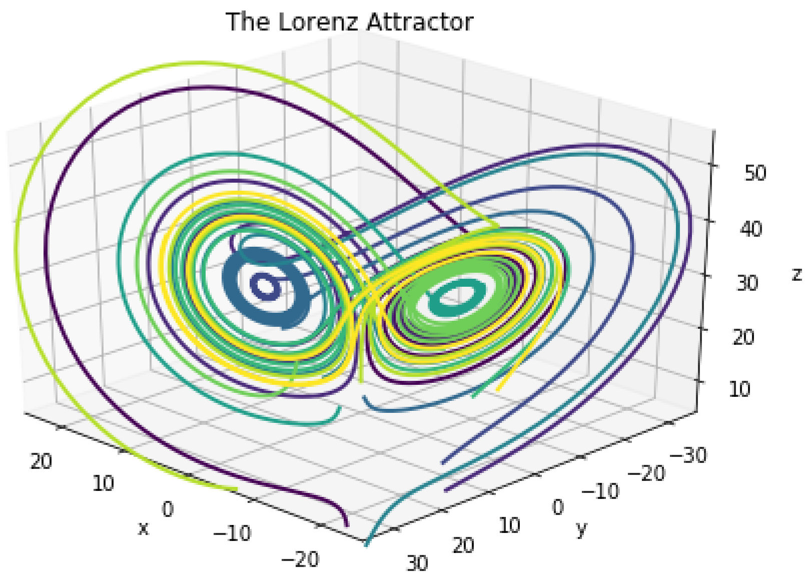


Figure 1 shows the Lorenz Attractor where σ = 10.0, ρ = 28.0, β = 8/3 and N = 10 computations done over 4 time steps.

Using the function, we also generate a time series over an interval [0:1000] and, integrating the differential equations once more, plot the output for 3 random starting points:

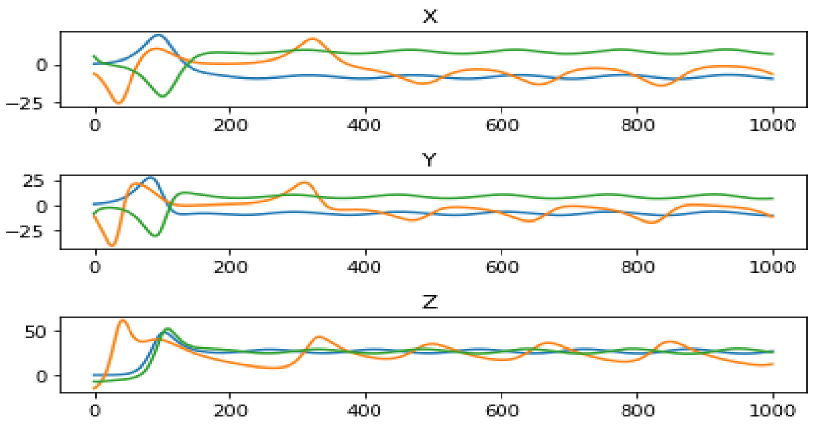


Figure 2 - Plot of output from solving 3 random starting points over a time interval [0:1000]

3 Building the Neural Network Models in Keras

Keras is a high level user-friendly Python library built on top of other powerful libraries such as Theano and TensorFlow. We fit two models, the LSTM network and the traditional RNN, to predict one-step ahead the state of the Lorenz System, utilizing the data generated by solving the system as above.

3.1 Convert Data into format for Keras

The data generated is then converted into a format that is usable by the neural network models. We split the data into two sets, the training set consisting of 70% of the data and the testing set consisting the remaining 30%.

```
Lorenz_train.shape:  (7, 1000, 3)
Lorenz_test.shape :  (3, 1000, 3)
```

The training dataset contains 7 sequences, each with 1000 data points that have 3 inputs (for x, y, and z). Likewise, the testing dataset has 3 sequences with the same format as the training dataset. However, the training and testing input for the neural network models needs to be in the form of: [*samples, time steps, features*]. We prepare both the training and testing data and reshape it into the expected structure:

```
Shape of training input:  (7912, 10, 3)
Shape of training output:  (7912, 3)
Shape of test input:  (3956, 10, 3)
Shape of testing output:  (3956, 3)
```

For both training and testing inputs, we use a sequence of size 10, 7912 and 3956 time steps/observations for the training and testing respectively, each with 3-dimensional format (x, y, z).

3.2 The Traditional Recurrent Neural Network (RNN)

RNNs are a type of neural network that utilize loops to persist information throughout the network, while processing sequential data one element at a time. They are trainable by specialized weight adaptation algorithms such as *Backpropagation Through Time* algorithm that implements a gradient descent based learning method [8]. However, the problem with most RNN models is that as the number of time steps increases, the backpropagation gradients either start vanishing or accumulating and exploding, affecting how the network learns the relationships between initial and later states of the system over a significant period of time [8, 9].

Our network architecture is as follows:

```
RNNmodel = Sequential()
RNNmodel.add(SimpleRNN(16, input_shape=(None, 3)))
RNNmodel.add(Dense(3))
RNNmodel.compile(loss='mean_squared_error', optimizer='adam')
```

Training the model:

```
RNNmodel.fit(x_train,y_train,validation_data=(x_test,y_test),verbose=2,
epochs=50)
```

Feeding the training and testing data into the model:

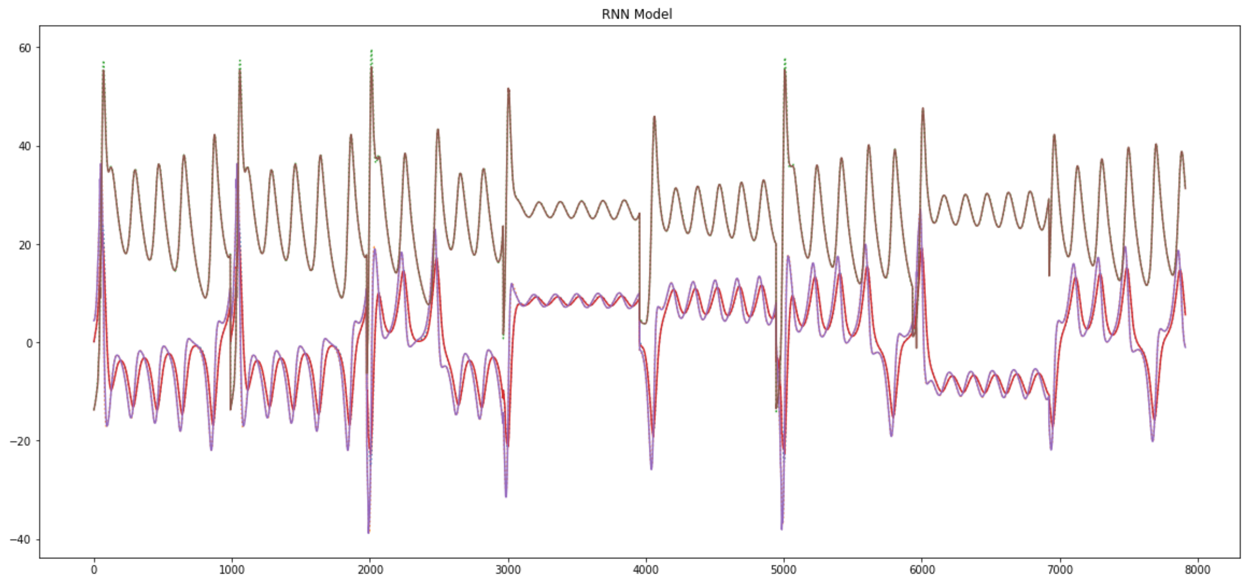


Figure 3: RNN Model predicting (x, y, z)

3.3 LSTM Neural Networks

One of the best ways to overcome the vanishing and/or exploding gradient problem is to use LSTM neural networks. These are a special kind of RNN, capable of learning long-term dependencies by remembering information for long periods of time. LSTM networks utilize a gated cell to preserve the error that can be backpropagated through time and layers, thus allowing the network to continue learning over many time steps [10, 11]. They are able to process and predict time series sequences without forgetting information about previous states of the system. They are very effective in learning long-term dependencies, which regular RNNs fail to learn efficiently as the system grows [12].

The architecture employed for the LSTM is as follows:

```
        model = Sequential()
        model.add(LSTM(16, input_shape=(None, 3)))
        model.add(Dense(3))
        model.compile(loss='mean_squared_error', optimizer='adam')
```

Training the model:

```
        model.fit(x_train,y_train,validation_data=(x_test,y_test),verbose=2,epo
        chs=50)
```

Feeding the training and testing data into the model:

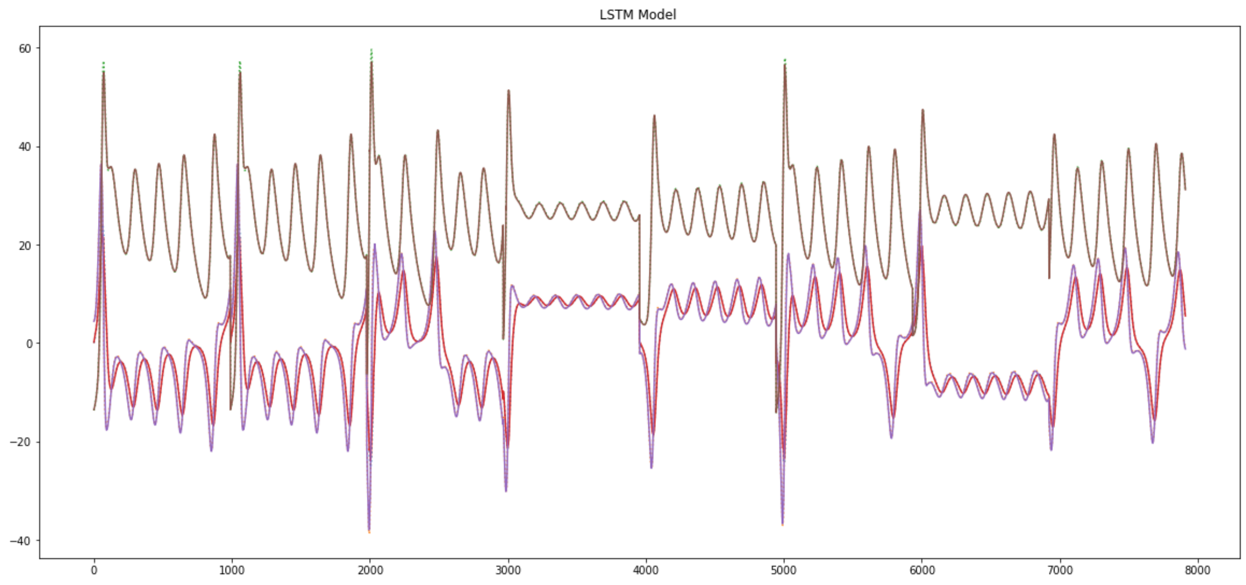


Figure 4: LSTM Model predicting (x, y, z)

4 Results

Both the traditional RNN and the LSTM were applied to chaotic data generated from the Lorenz System. Results for the network training and evaluation of the network's predictive capabilities on the data are described below.

Training both networks on 7912 samples and validating on 3956 samples:

Model	Epochs	Validation Loss
RNN	50	0.2149
LSTM	50	0.1523
RNN	75	0.0559
LSTM	75	0.0202

Table 1: Performance comparison for different epochs on testing prediction of the two models

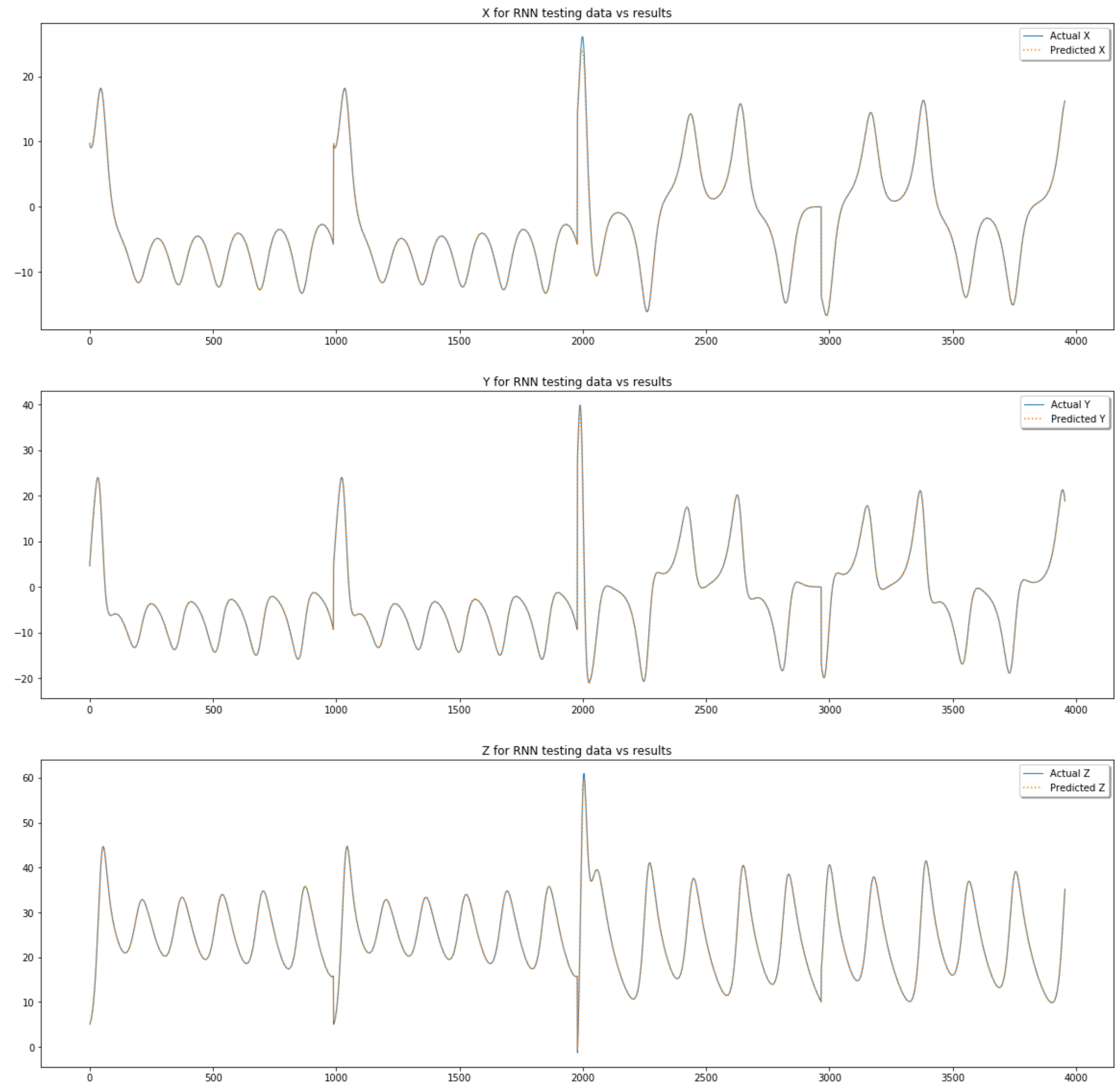
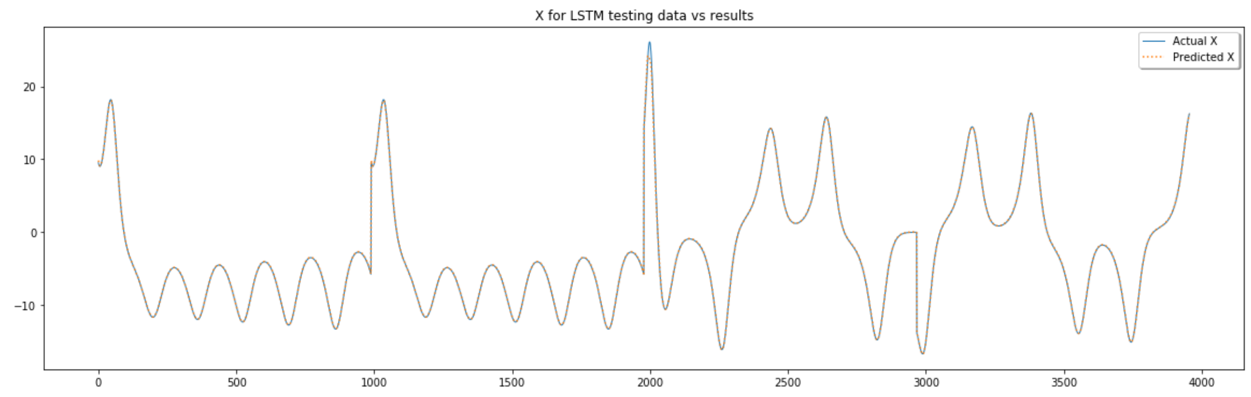
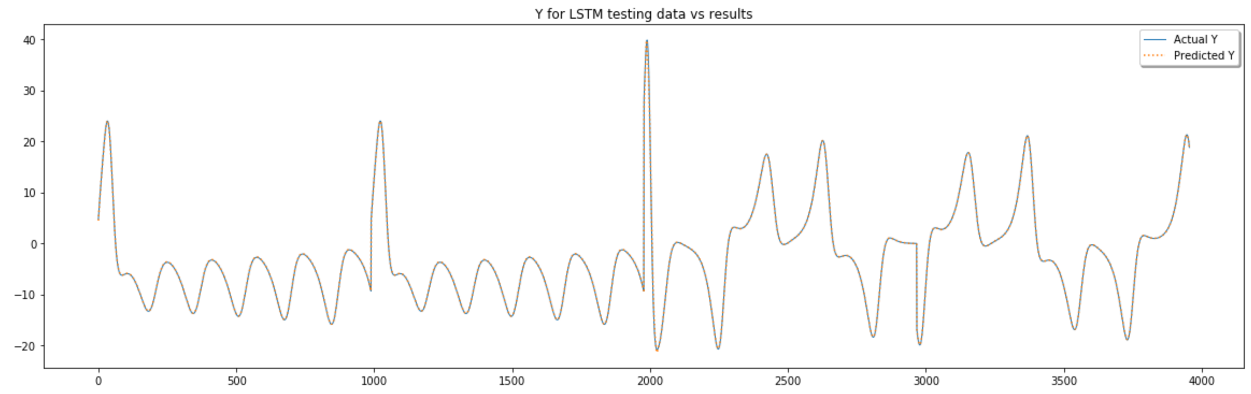


Figure 5 shows the Lorenz chaotic time series that were used in the present application for training and testing of a traditional RNN.





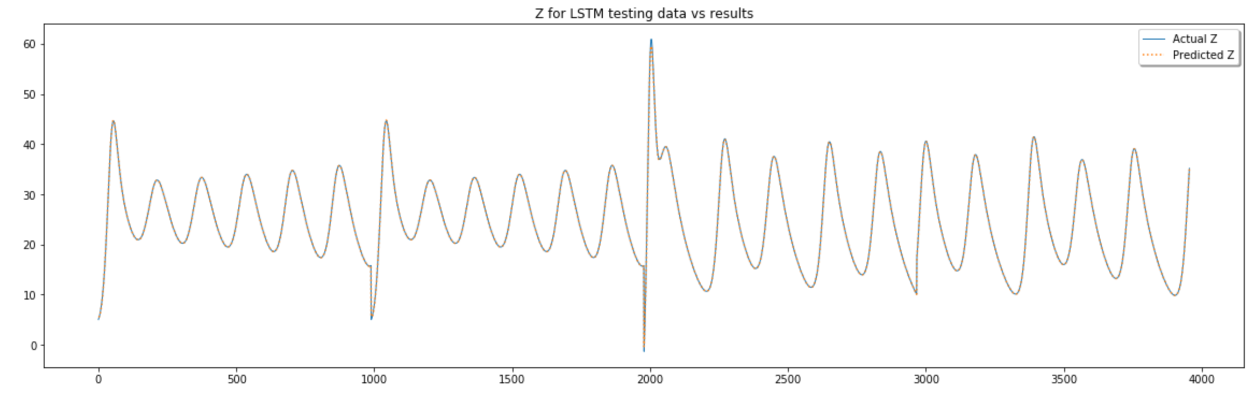


Figure 6 shows the Lorenz chaotic time series that were used in the present application for training and testing of a LSTM network.

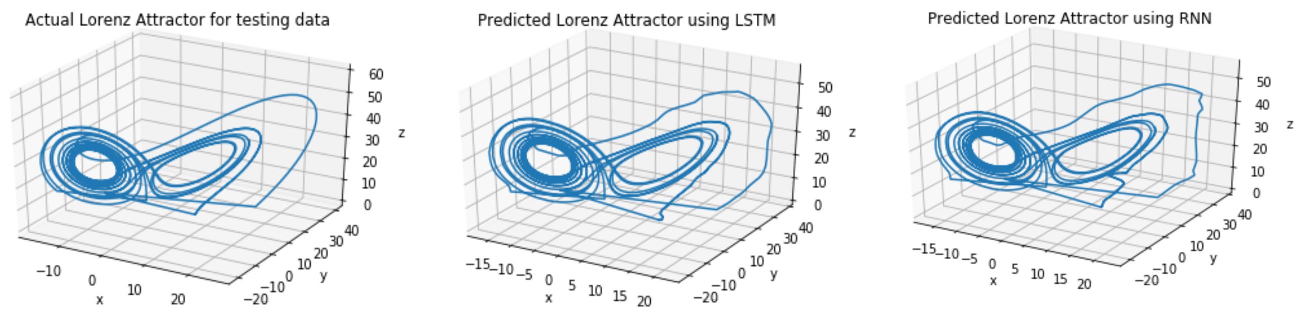


Figure 7: Comparison of the two models on test data prediction.

The two models do a relatively good job in predicting the outcome of the Lorenz System over time. The results showed that on average, LSTM performs better predictions than traditional RNN in systems with chaotic input.

4.1 Testing with single starting point: predicting 1-step ahead

To further learn the efficiency of LSTM in predicting, we also analyzed and compared the two models ability to predict one-step ahead, based on a single starting point.

```
    Shape of data:  (1, 1000, 3)
    Shape of windowed data:  (1978, 10, 3)
```

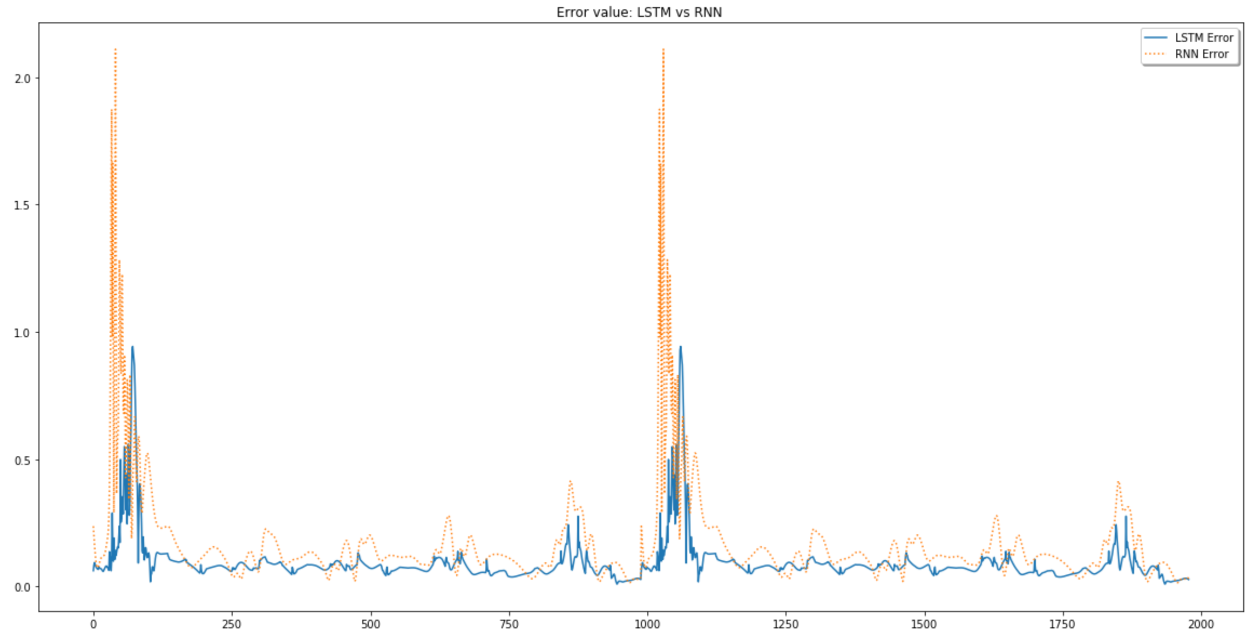


Figure 8: One-step prediction performance on test data. Graph of true test output against predicted output for both LSTM and RNN.

The performance of one-step ahead prediction is shown in the following table.

	RNN	**LSTM**
Average Prediction Error	0.1691547258621755	0.09410667635011637

Table 2: Performance Comparison for One-step Ahead Prediction

RNN

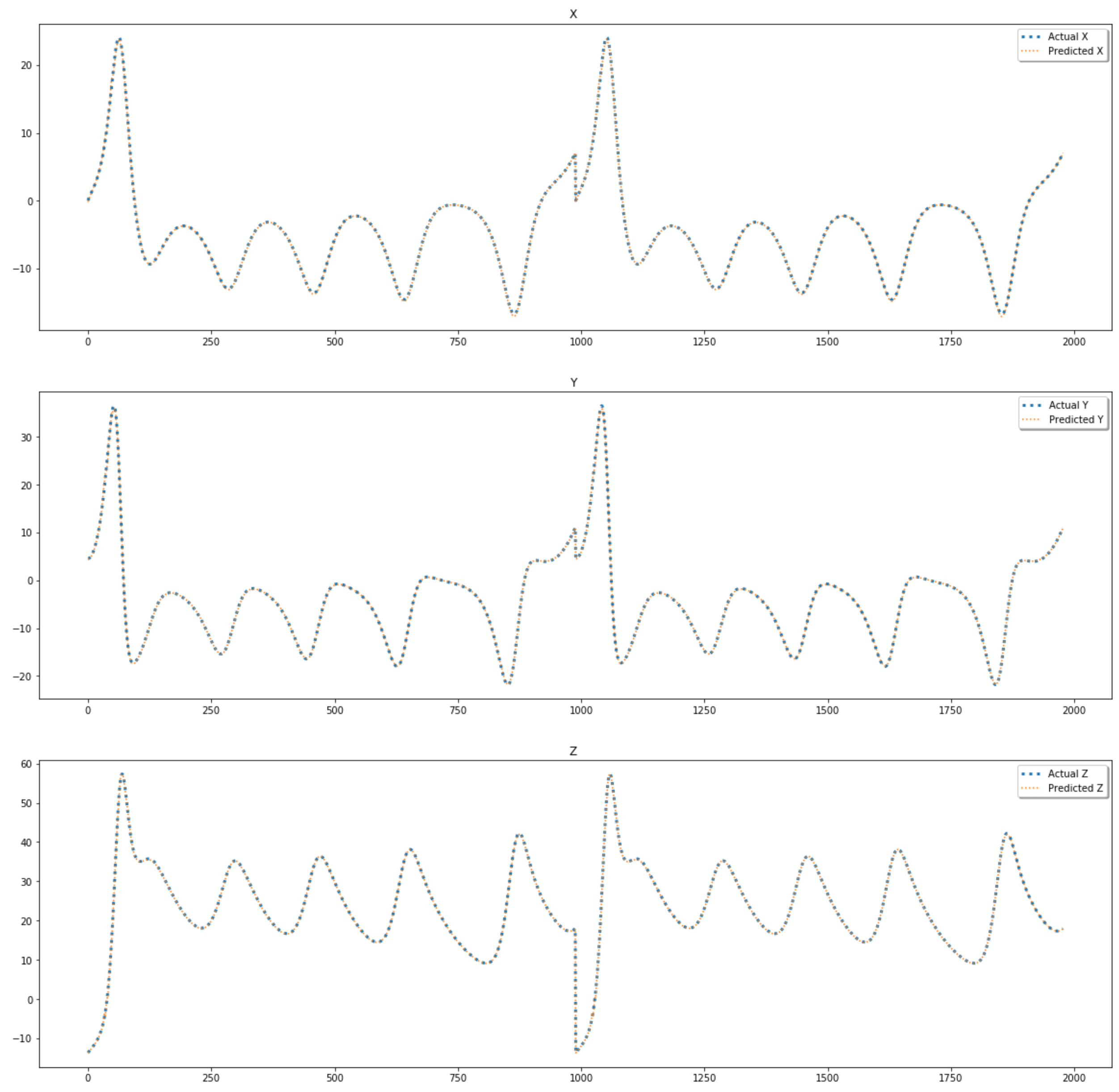


Figure 9: Graph showing RNN true test output against predicted output based on a single starting point.

LSTM

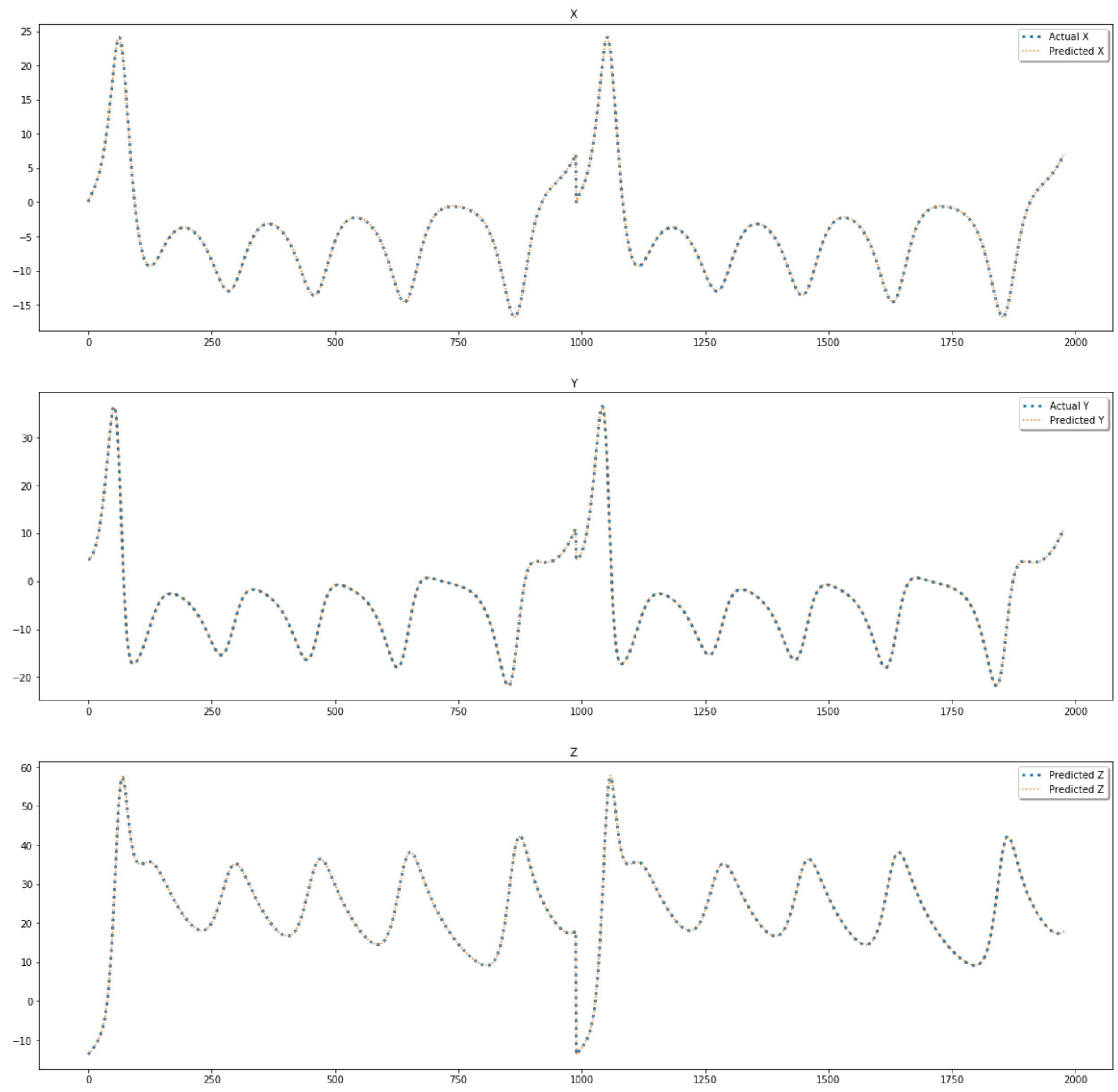


Figure 10: Graph showing LSTM true test output against predicted output based on a single starting point.

5 Conclusion

In this paper, a traditional RNN and a LSTM network are used in the context of modeling a chaotic system and predicting its state, one-step ahead. It is observed that the LSTM network outperforms the traditional RNN due to its ability to learn long term dependencies as the system behaves grows. Learning about the chaotic systems can give useful insights in weather forecasting, traffic, and stock market predictions. As for future work, it would be interesting to explore other types of NNs and investigate their effectiveness in predicting chaotic systems in comparison to the traditional RNN and the LSTM. Another area of exploration would be implementing RNN and LSTM models for multi-step time series forecasting and comparing the results as was done for the single-step prediction in this paper.

6 Acknowledgements

Special thanks to Dr. Thomas Gibbons for his support and guidance throughout the course of this independent study project. Many thanks to Dr. Luther Qson for the resources and textbooks on Chaos and Dynamic Systems and to the teams behind Keras and Jupyter Notebooks for making it easy for us to implement and run our model.

References

1. Lorenz, Edward N. "Deterministic Nonperiodic Flow". *Journal of the Atmospheric Sciences,* vol.20, 130-141, 1963.
2. Banks, J., Brooks, J., Cairns, G., Davis, G., and Stacy, P. (1992), "On Devaney's definition of chaos," Amer. Math. Monthly 99; 332-334
3. Devaney, R. L. (1989), An Introduction to Chaotic Dynamical Systems, (AddisonWesley, Redwood City)
4. Woolley, Jonathan W., Agarwal, P. K., and Baker, John. Modeling and prediction of chaotic systems with artificial neural networks. *International Journal for Numerical Methods in Fluids.* 63:989–1004, 2010.
5. Sanjay Vasant Dudul. Prediction of a Lorenz chaotic attractor using two-layer perceptron neural network. *Applied Soft Computing 5 (2005) 333–355*.
6. Daniel Hsu. (2017). Time Series Forecasting Based on Augmented Long Short-Term Memory. Retrieved from https://arxiv.org/pdf/1707.00666.pdf
7. Project Jupyter (2017). *Exploring the Lorenz System of Differential Equations.* Retrieved from http://ipywidgets.readthedocs.io/en/latest/examples/Lorenz%20Differential%20Equations.html
8. Lipton, Z.C., Berkowitz, J., Elkan, C.: A Critical Review of Recurrent Neural Networks for Sequence Learning. arXiv preprint arXiv:1506.00019 (2015) https://arxiv.org/pdf/1506.00019.pdf
9. Hochreiter, S.: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6(02) (1998) 107–116
10. Andy Thomas. 2017. Recurrent neural networks and LSTM tutorial in Python and TensorFlow. Retrieved from http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/
11. Christopher Olah. 2015. Understanding LSTM Networks. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

12. Zaytar, M.A. Amrani, C.E. Sequence to Sequence Weather Forecasting with Long Short-Term Memory Recurrent Neural Networks. *International Journal of Computer Applications (0975 - 8887). Volume 143 - No.11, June 2016*. Retrieved from https://pdfs.semanticscholar.org/f9ae/308836dc0f96325671be6d75c38a97f42a8b.pdf