MAST 680 - Assignment 3

Kylian Ajavon

April 10, 2023

Abstract

We investigate the use of neural networks for predicting the next state of a point in the Lorenz system. Our results show that the neural network performs well for certain values of the system parameter ρ, but does not perform well when it comes to predicting the transition to a different lobe in the phase space when $\rho = 28$. We discuss the limitations of our approach and highlight the challenges of using neural networks to model chaotic systems.

1 Introduction and Overview

Predicting the behavior of complex systems is a fundamental challenge in many areas of science and engineering. This is especially true when a system is chaotic, meaning that its behavior is highly sensitive to its initial conditions and parameters, and it is difficult to predict using traditional methods. Our goal in this assignment will be to train a neural network to predict the next state of a dynamical system. The neural network will take as inputs the state variables (x, y, z) of the system at some time t, along with one of the parameters of the system. The output of the neural network will be the state variables at time $t + \Delta t$, for some small $\Delta t > 0$. The dynamical system we will use is the Lorenz system, with some given values of one of its parameters. After finding adequate hyperparameters for our neural network, we will measure the performance of our model by making various predictions: first, we will predict the next state of the system, and then we will predict how far into the future the system will go from one "lobe" to the other.

2 Theoretical Background

2.1 Lorenz Equations

The Lorenz system is a set of three ordinary differential equations, first studied by Edward Lorenz as a way to study the behavior of a simplified model of the Earth's atmosphere. The system is given by:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \tag{1}$$

where σ, ρ and β are the system parameters.

The behavior of the Lorenz system is known to change significantly when the parameter ρ takes on certain values. The values $\rho = 10$, $\rho = 28$, and $\rho = 40$ are of particular interest. When $\rho = 10$, the Lorenz system has a single stable fixed point at the origin. This means that for most initial conditions, the system will converge to this fixed point and remain there. When $\rho = 28$, the system exhibits chaotic behavior, with a strange attractor in the shape of a butterfly. This means that small perturbations to the initial conditions can lead to vastly different outcomes, and the trajectory of the system in phase space is nonperiodic. When $\rho = 40$, the system undergoes a bifurcation, in which the single stable fixed point at the origin becomes unstable, and two new stable fixed points appear. This means that the behavior of the system can switch between two different attractors, depending on the initial conditions.

2.2 Neural Networks

Neural networks are a type of machine learning algorithm inspired by the structure and function of the human brain. They are composed of nodes, called neurons, which receive inputs, perform a computation, and produce an output. These nodes are organized into layers, with each layer typically consisting of multiple neurons that perform a similar type of computation. Each layer of a neural network works by applying a linear transformation and a non-linear function to the input. Given an input $x \in \mathbb{R}^n$, one layer of a neural network is defined by

$$y = \sigma(Wx + b)$$

where $\sigma(.)$ represents a nonlinear **activation function**, W is a matrix of **weights** and b is a vector of **biases**. The output y of that layer then becomes the input for the next layer, where the same operations are applied. The output at the end of the neural network is therefore a succession of linear transformations and non-linear activation functions on the original input, i.e.

$$y = \sigma(W_k\sigma(W_{k-1}\sigma(...\sigma(W_1x + b_1)...) + b_{k-1}) + b_k).$$

The output y is then used to compute a **loss function** $\mathcal{L}(y, W, b)$, which evaluates the performance (or the error) of the neural network based on the training data. We wish to minimize that error, or equivalently solve the minimization problem where we find the minimum loss with respect to the weights and biases. Therefore, the weights and biases are trainable parameters, that are adjusted in the process of **backpropagation**. After each round of backpropagation, the weights and biases will be adjusted to a point where the neural network is a good interpolator for the sample points in our training data.

3 Algorithm Implementation and Development

3.1 Data generation & processing

First, the data representing the Lorenz system (for given initial conditions) had to be generated. This was done with the function `generate_points` (see Appendix A), which takes as input some initial conditions (x_0, y_0, z_0), a parameter ρ and `tmax` set to 10000 by default. This last argument determines the step size of in time, i.e. the quantity Δt. As we are performing a machine learning task, we also had to split our generated data into a training set and a testing set. The splitting was performed using the `split_data` (see Appendix A). The process of generating and splitting the data was done for $\rho = 10, 28, 40$. For each ρ, the training and testing sets were then concatenated to obtain unified training and testing sets.

3.2 Neural networks implementation

Our neural networks implementation are done by using the Python library `tensorflow`. We created functions to compare the performance of neural networks using different loss functions (see `compare_loss_functions` in Appendix A), to compute the relative errors of the model's predictions (see `compare_predictions_plot`), etc. These functions were all used using features of `tensorflow`, which simplifies the process of creating and training neural networkks, as well as making predictions with them.

4 Computational Results

4.1 Loss functions

After generating our points (with the initial conditions [0,1,1.05]) and splitting the data into training and testing sets, we trained four neural networks with four different loss functions on the exact same training dataset. The four loss functions that were chosen are the MSE (Mean Squared Error), MAE (Mean Absolute Error), Huber and LogCosh loss functions. From [1], we can see that after 100 epochs, the Huber loss function has the best performance.

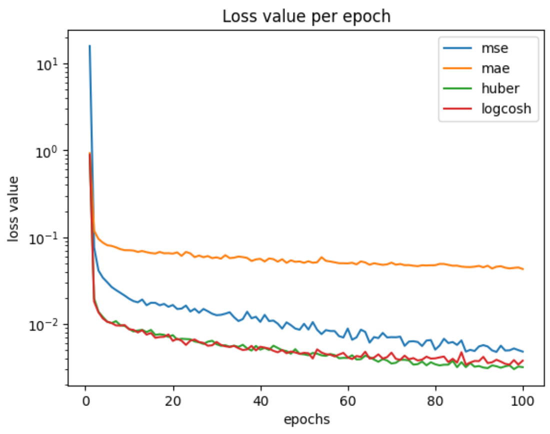
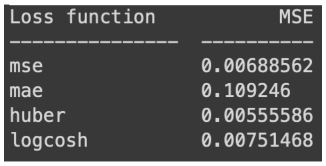


Figure 1: Loss value during training with 4 different loss functions

This is further confirmed by using a validation set within the training set, and by evaluating the performance of each model's predictions. Here, [2] shows that the Huber loss function gives the predictions with the smallest error, out of all the loss functions we tested.



```
Loss function           MSE
---------------  ----------
mse              0.00688562
mae              0.109246
huber            0.00555586
logcosh          0.00751468
```

Figure 2: Performance of loss functions based on predictions on validation set

4.2 Predictions on known ρ

From this point forward, we use a single neural network trained with the Huber loss function. We wish to see how long the network's predictions can stay accurate, by successively feeding back the output of the network as input. As we can see, the predictions diverge from true value quickly, as the relative error increases exponentially. Despite the oscillations, the errors tend to increase over time, as expected.

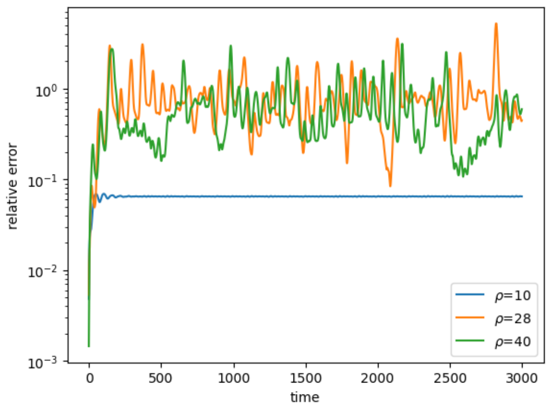
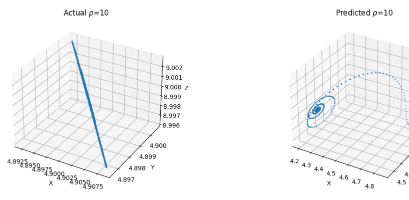
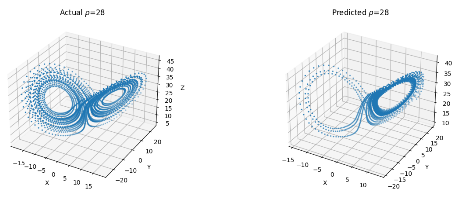


Figure 3: Relative error of successive predictions of the model

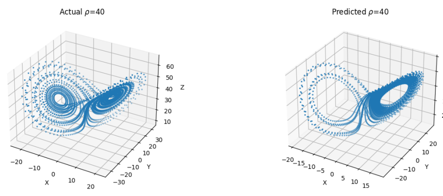
Overall, when the input has a value of ρ that was in the training data, the model's predictions are decent, especially for $\rho = 28$ and $\rho = 40$. For $\rho = 10$, the results are not as good, mainly due to the fact that in the actual data, the system converges to a single stable fixed point, which the model is not able to predict.



(a) $\rho = 10$



(b) $\rho = 28$

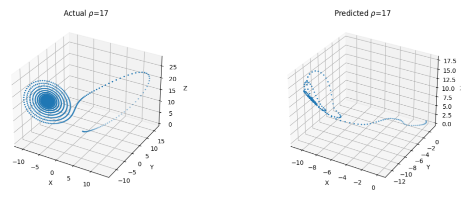


(c) $\rho = 40$

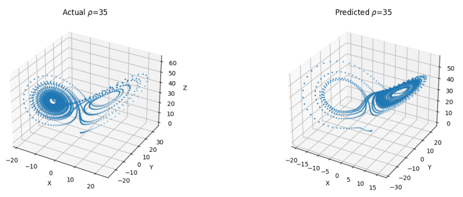
Figure 4: Actual and predicted positions of state variables for a given ρ

4.3 Predictions on unknown ρ

We reproduce the same experiment as above, but this time we use values of ρ on which the model was not trained. As expected, the predictions are not very good. However, the model is (surprisingly) able to determine the general behavior of the system without prior knowledge of how ρ would impact it. For $\rho = 17$, the model was able to determine that the system starts with a "tail", then oscillates and converges to a single fixed points. For $\rho = 35$, the model was able to determine that the system would have a similar behavior as $\rho = 40$.



(a) $\rho = 17$



(b) $\rho = 35$

Figure 5: Actual and predicted positions of state variables for a given (unknown) ρ

4.4 Transition prediction

The last task of the assignment was to train a neural network to identify when the transition to another lobe is imminent, after fixing $\rho = 28$. We used a neural network with 8 hidden layers and using the Huber loss function when training the model. As shown by the loss values in [6], the model performs poorly. There was no improvement in the results by changing some hyperparameters such as the depth of the model, the learning rate, the loss function, etc.

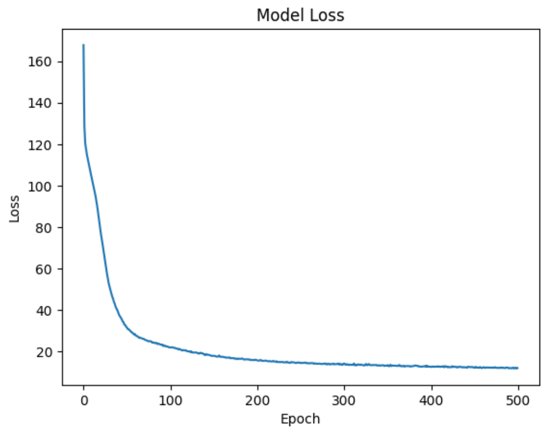


Figure 6: Evolution of loss value of the model during training

5 Summary and Conclusions

In this project, we explored the use of neural networks for predicting the behavior of a point in the Lorenz system. Our results show that the neural network can somewhat accurately predict the next state of the system for some values of ρ, but only for the first few steps in time. Moreover, it struggles to capture the complex transitions between different lobes when $\rho = 28$. These results highlight the limitations of using neural networks for modelling chaotic systems, and suggest the need for more robust and adaptive approaches for this particular task. Overall, this assignment provided insights into the potential and the challenges of using neural networks for predicting the behavior of dynamical systems.