

### Assignment 3

#### Forecasting Chaos with Neural Networks

DUE: Friday, April 7, 2023 at 11:59 pm PST  
Late assignments will **NOT** be accepted

Professor: Jason J. Bramburger

---

The Lorenz equations are possibly the most famous system of ordinary differential equations that exhibit chaotic dynamics. They are given by

$$\begin{aligned}\frac{dx}{dt} &= 10(y - x) \\ \frac{dy}{dt} &= x(\rho - z) - y \\ \frac{dz}{dt} &= xy - \frac{8}{3}z\end{aligned}\tag{1}$$

where  $\rho > 0$  is a system parameter. Some background information, figures, and even code snippets for the Lorenz equations are given on the [wikipedia page](#).

Fix a small  $\Delta t > 0$ . Your task is to train a neural network to take solutions

$$(x(t), y(t), z(t))$$

to

$$(x(t + \Delta t), y(t + \Delta t), z(t + \Delta t))$$

for  $\rho = 10, 28$ , and  $40$ . You want to only train one neural network that works for all three parameter values, so your input will be four-dimensional: 3 state variables  $(x, y, z)$  and 1 parameter value  $\rho$ , with the output being the same. Passing data through the network should not change the parameter value  $\rho$  fed into it, but its value will influence how the state variables are advanced. Experiment with the following:

1. The loss function that provides the best predictions.
2. By successively feeding the output of the network back into the network we can advance the predictions  $\Delta t$  time units each iteration. How far into the future do your predictions remain accurate at the trained parameter values  $\rho = 10, 28$ , and  $40$ ? What happens if you use a bigger or smaller  $\Delta t$ ?
3. How well does your trained neural network predict the future state for parameter values outside the training data? Try  $\rho = 17$  and  $\rho = 35$ .

After completing the above, fix  $\rho = 28$  and train a neural network to identify when a transition from one lobe to another is imminent. Determine how far in advance you can make this prediction. Note: you will have to label the transitions in a test set in order to do this task.

**Comment:** You may generate your training data using whatever method is easiest to you. That is, using a built-in ODE integrator such as `ode23/ode45` in MATLAB or `odeint` in `scipy` is perfectly acceptable (and probably more accurate). If you are going to simulate the manually, you must at least use a [Runge-Kutta](#) scheme - Euler stepping is not acceptable as it will be either too inaccurate or too slow.