AERSP 497 HW3 - Neural Networks for System Identification

Vivek Mathew

1 Introduction

This report outlines the approach and findings from implementing a single-hidden-layer feedforward neural network to model the mapping between a system's state vector and its time derivatives. The neural network was trained on the same dataset as in HW1, where `a` represents the state vector (x, y, z) and `da` represents its corresponding time derivatives $(\dot{x}, \dot{y}, \dot{z})$. The data was generated from the Lorenz system, a classic example of a chaotic dynamical system whose attractor is shown in Figure 1.

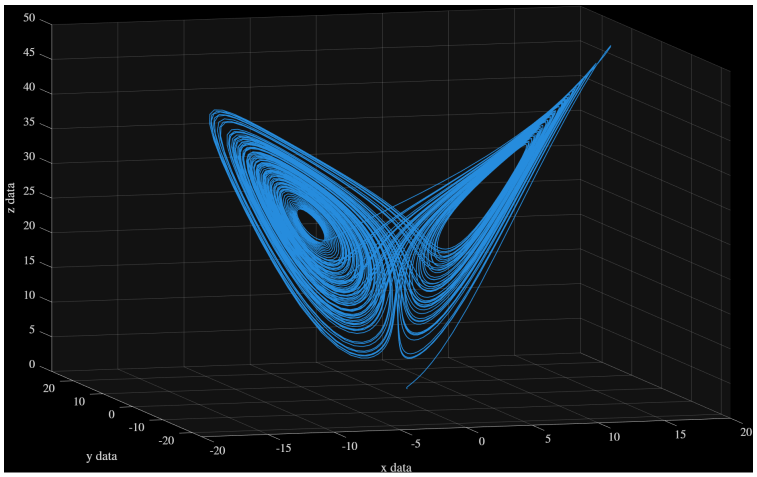

Figure 1: The Lorenz system attractor, representing the dynamics the neural network is intended to learn.

The objective was to investigate how a neural network compares to linear regression methods in capturing the nonlinear dynamics of the system, as well as to evaluate the effects of feature scaling, feature dimensionality, and network size on test accuracy.

2 Methods and Implementation

2.1 Data Preparation

As in HW1, the dataset was randomly split into:

- 70% training data
- 30% test data

No random number generator seed was set, meaning that each execution of the code results in slightly different train/test splits and randomized weight initializations. Consequently, results such as test error vary slightly between runs. Nonetheless, across repeated experiments the test error consistently fell within the range of 12.5–13%.

A second-order polynomial feature expansion was applied to the input states (x, y, z), resulting in 9 features (linear, quadratic, and pairwise interaction terms). Min–max scaling was then applied to normalize each feature to the range $[0, 1]$. This step was crucial for stable and efficient training, as explained later.

2.2 Neural Network Architecture

The neural network used in this study consisted of:

- Input layer: 9 features
- Hidden layer: between 20 to 200 neurons tested; final best-performing model used 50
- Output layer: 3 neurons corresponding to $(\dot{x}, \dot{y}, \dot{z})$

The hidden layer employed the sigmoid activation function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

while the output layer was linear, as appropriate for regression.

Weights were initialized randomly from a Gaussian distribution with small variance. Training was performed using batch gradient descent with backpropagation, minimizing the mean squared error (MSE) loss function:

$$J = \frac{1}{m} \sum_{i=1}^{m} \|\hat{y}^{(i)} - y^{(i)}\|^2$$

The learning rate α was set to 0.05, which provided stable convergence and the lowest observed test error.

3 Results and Discussion

3.1 Effect of Feature Scaling

Without min–max normalization, the network consistently failed to converge to a low test error. This is because features with larger numeric ranges dominated the weight updates, leading to unstable gradients and poor optimization. With min–max scaling, all features contributed comparably to the learning process, and the test error decreased significantly. Thus, feature scaling was essential for the model's success.

3.2 Effect of Number of Features

For a fixed hidden layer size, increasing the number of input features improved the ability of the network to capture nonlinear relationships in the data. However, more features also increased the number of trainable parameters, which can raise the risk of overfitting if not paired with sufficient data. The second-order expansion (9 features) provided a good balance, yielding lower test error compared to linear-only features.

3.3 Effect of Network Size

For a fixed feature set, increasing the number of neurons in the hidden layer improved accuracy up to a point. Too few neurons limited the expressiveness of the network, while too many led to diminishing returns in performance and longer training times. Empirically, I found that a hidden layer size of 50 neurons gave the lowest test error.

The learning rate also played a critical role. After testing multiple values, $\alpha = 0.05$ consistently minimized the test error compared to smaller or larger values, which either slowed training or caused divergence. The training process itself was monitored by tracking the MSE at each epoch. Figure 2 shows the training loss curve, which demonstrates a rapid initial decrease followed by stable convergence, validating the choice of learning rate.

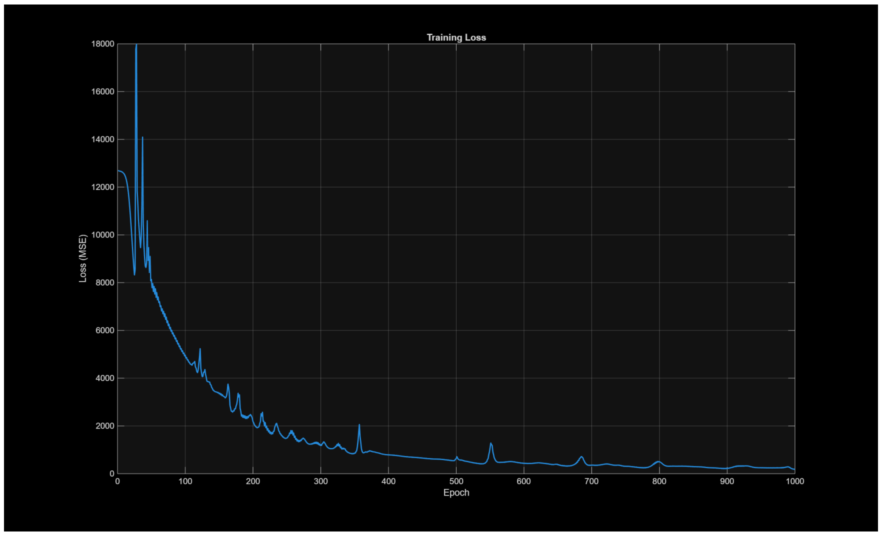

Figure 2: Training loss (Mean Squared Error) plotted against the number of epochs, showing stable convergence.

3.4 Observed Test Error

Across multiple runs, the normalized test error was consistently found in the range of 12.5–13%. This value reflects both the effectiveness of the chosen architecture and the stochastic nature of neural network training due to random initialization. Runs with different random seeds occasionally produced slightly higher or lower errors, but the 12.5–13% range was stable and repeatable.