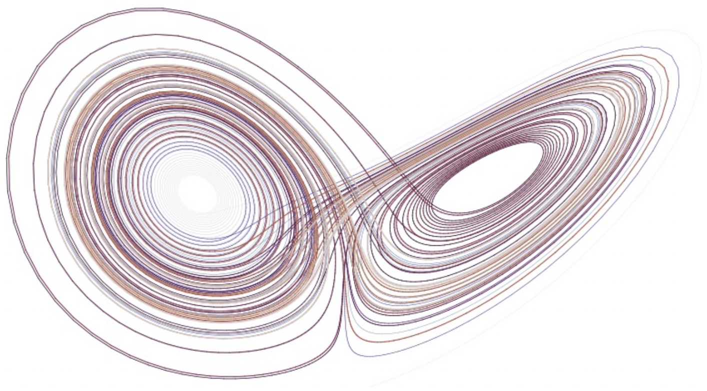
Understanding Neural Networks for Predicting Chaotic Dynamics

Sebastian M.D.

May 2024



Abstract

This article explains my research on using different neural network architectures to predict chaotic dynamics, specifically focusing on the Lorenz 63 system. Three neural network architectures were compared: LSTM Recurrent Neural Networks, Echo State Networks, and Transformers. By measuring the Mean Squared Error (MSE) of their predictions, we determine which architecture is most effective for this task.

1 Introduction

Chaotic systems are deterministic systems which are highly sensitive to initial conditions, making long term predictions challenging. The weather is usually attributed as such a chaotic system, incredibly hard to predict in the long term. One such system aimed to model the weather's atmospheric convection is called the Lorenz 63 system, envisioned by Edward Lorenz in 1963. Traditionally, numerical methods have been used to predict these systems, but recent research has explored neural networks as an alternative solution. In this study, different neural network architectures are tested on their accuracy in predicting the Lorenz 63 system. The goal of the study is to determine which neural network architecture is most promising in predicting chaotic dynamics.

2 Theory

2.1 The Lorenz System

The Lorenz system is a set of three mathematical equations that were developed by Edward Lorenz to study weather patterns. They are now famous for showing how small changes can lead to very different outcomes. The equations are as following:

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

In these equations, x, y, and z are variables that change over time, and σ, ρ, and β are constants with the following values:

$$\sigma = 10, \quad \rho = 28, \quad \beta = \frac{8}{3}$$

These equations are differential equations, which are equations that relate a variable to it's rate of change, it's so called derivative. The derivatives: $\frac{dx}{dt}$, $\frac{dy}{dt}$, $\frac{dz}{dt}$ are directly dependent on the variables x, y, z which makes them differential equations.

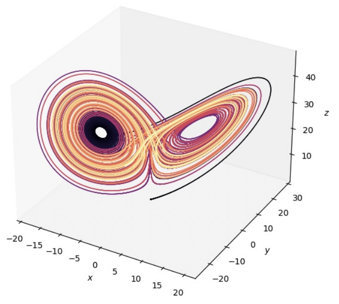


Figure 1: The Lorenz attractor visualized with the RK4 method for 100 time units

When we use these values, the equations create a pattern called the Lorenz attractor, see figure 1. This pattern looks like a butterfly's wings and shows how the system evolves over time. Even though the system is determinstic, its behavior is highly sensitive to initial conditions, meaning that a small change in initial starting values x, y, z, can lead to radically different x, y, z values after a certain t timesteps. This is why the Lorenz system is a great example of chaos theory.

2.2 Neural Networks

Neural networks are a type of machine learning model inspired by how the human brain works. They consist of layers of interconnected nodes, which act like neurons, processing input data to produce an output. The connections between these nodes have weights and biases, which are parameters that decide what will get passed to the next node. The weights and biases are adjusted during the learning process.

At the core of neural networks is a process called feedforward computation, where data moves from one layer of nodes to the next. For example, if we have an input vector(a list of inputs) $\vec{x}$, the output vector(list of outputs) $\vec{y}$ is calculated as follows:

$$\vec{y} = f(W\vec{x} + \vec{b})$$

Here W is the weights matrix, a 2-dimensional list of numbers that hold the weights that are to be multiplied with the list of inputs, and $\vec{b}$ is the bias vector, containing a list of numbers that will be added to the result after multiplication with the weights matrix. Finally, f is the activation function, which will map the output into something reasonable, for example numbers between 0-1.

To adjust the weights and biases, the network is trained on data by starting out with random weights and biases, then by measuring the error in prediction it will make small adjustment to these parameters that minimize the error. To measure the error in predictions, a cost function is used. The Mean Squared Error (MSE) is commonly used in regression tasks. By adjusting the weights in this way, the neural network learns to make accurate predictions on new, unseen data over time.

2.3 RNNs

A Recurrent Neural Network (RNN) is a type of neural network designed to recognize patterns in sequences of data. Unlike traditional neural networks, RNNs have connections that form loops. This allows them to maintain a 'memory' of previous inputs, which helps in processing sequential data like the Lorenz system.

In an RNN, each step in the sequence has a hidden state that captures information from previous steps. When an input is fed into the RNN, it updates its hidden state and produces an output. This hidden state is then passed to the next step along with the next input, enabling the network to remember past information.

The RNN processes updates the hidden state as follows:

$$h^{(t)} = \tanh(Wh^{(t-1)} + Ux^{(t)} + b_h) \tag{1}$$

Here, $h^{(t)}$ is the hidden state at time step t. The weights W, U, and V, along with biases b_h and b_y, are parameters that the network learns during training. These parameters are the same across all time steps, allowing the RNN to apply the same rules to each part of the sequence.

Long Short-Term Memory (LSTM) networks are an improved RNN architecture that is used in this study, LSTMs use special units called gates to control the flow of information and improve long term memory.

2.4 Transformers

Transformers are a type of neural network introduced in the paper "Attention is All You Need". Unlike previous models, they don't read data step-by-step but process the whole sequence at once. This allows them to work faster and handle more complex tasks.

The key idea in Transformers is the self-attention mechanism. Each word in a sentence is turned into three vectors:

- Query (Q): What the word is looking for
- Key (K): What the word offers
- Value (V): The information the word contains

The self-attention mechanism compares these vectors to find out how much each word should pay attention to the others. For example, in the sentence "The cat sat on the mat," the word "cat" would pay more attention to "mat" than to "the".

To make this comparison, the mechanism calculates:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

This formula helps the model learn the relationships between words in a sentence. The attention mechanism is usually applied in parallel and in layers. These layers are repeated in blocks to form the full transformer network.

2.5 Reservoir Computing

Reservoir Computing (RC) is a type of Recurrent Neural Network (RNN) that is especially good at predicting chaotic systems. The main idea is that only part of the network is trained, making it faster and solving some problems found in traditional RNNs.

Echo State Networks (ESNs) are a popular example of RC. In ESNs, the network has a random, sparsely connected part called the reservoir. The reservoir's state at time t, $x^{(t)}$, is updated based on the previous state $x^{(t-1)}$, the input $u^{(t)}$, and the feedback from the previous output $y^{(t-1)}$. The equation is:

$$x^{(t)} = \tanh(\mathbf{W}x^{(t-1)} + \mathbf{W_{in}}u^{(t)} + \mathbf{W_{fb}}y^{(t-1)} + b)$$

Here, $\mathbf{W}$, $\mathbf{W_{in}}$, and $\mathbf{W_{fb}}$ are randomly generated matrices that are not trained.

The output $y^{(t)}$ is calculated using the trained weights $\mathbf{W_{out}}$:

$$y^{(t)} = \mathbf{W_{out}}x^{(t)}$$

The reservoir projects the input into a higher-dimensional space, making it easier to separate parts of the input. Only the output layer is trained, typically using a method like ridge regression, to minimize the error between the predicted and actual outputs.

3 Method

This project involves creating, training, and evaulating machine learning models using Python and the PyTorch library. The model architectures used are Transformers, LSTM RNN, and Echo State Network (ESN). The project's code is available on GitHub at: `https://github.com/SebCodesTheWeb/lorents-net`.

3.1 Generating Data

Data was generated using a numerical method called the RK4 method. This method simulates the Lorenz system with random intial positions, and then records and saves the positions of the system as it moves through time in a CSV file. This data was then used to train the neural networks.

3.2 Setting up the models

The LSTM RNN model was set up using PyTorch's inbuilt nn.LSTM class. The Adam optimizer was used to train the parameters. The Transformers model was also built using PyTorch and trained similiary to the LSTM RNN model. The ESN model was implemented using the ReservoirPY library and was trained with inbuilt ridge regression optimizer from ReservoirPy.

3.3 Hyperparameter Optimization

Hyperparameters are parameters that control the training process of a neural network. Hyperparameters could decide things like the number of layers, learning rate, training time, and network size of the model. A library called Optuna was used to find the best hyperparameters for the Transformers and LSTM RNN models. This process involves running multiple trials and selecting the best combination of parameters, it's computationally heavy and was trained for hundreds of combinations on four RTX 4090 GPUs. The ESN model was manually optimized due to bad library support with Optuna.

Once the models were trained they were then evaluated by predicting 500 time steps of the Lorenz system and comparing the predictions to the RK4 method. To evaluate the models, their Lorenz attractor shape, trajectory and MSE was plotted.

4 Results

4.1 Lorenz attractor

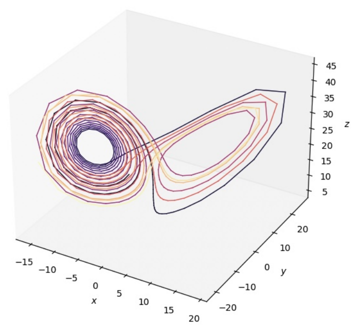


Figure 2: LSTM RNN

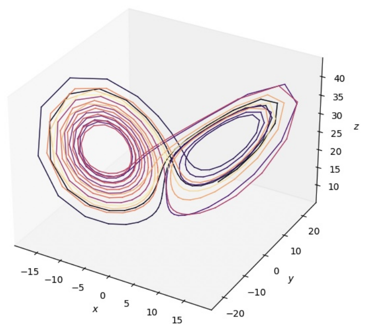


Figure 3: ESN

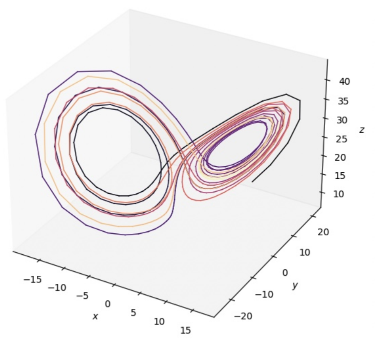


Figure 4: Transformers

4.2 Trajectory

View figures 5, 6 and 7 in Appendix A.

4.3 MSE

The average MSE for the LSTM RNN was 50.1579.
The average MSE for the ESN was 48.7797.
The average MSE for the Transformers network was 29.1590.

5 Discussion

The figures show that all models have learned the Lorenz patterns, producing attractors with two wings around two focal points. However, there are differences in accuracy. The Transformers model has the most accurate angle between the wings, but the proportions are off. The ESN model has better proportions between the two rings(which are supposed to be of close to equal radius). Overall, the ESN model appears to have the most accurate Lorenz attractor shape.

The trajectory plots indicate different levels of accuracy. The LSTM RNN deviates from the true path around timestep 40-50. The ESN model diverges around timestep 60-100. The Transformers model divererges around timestep 100-120.

Considering the input sequence lengths, the ESN can predict accurately for 70-80 timesteps, the LSTM RNN for 30-40 timesteps, and the Transformers model for about 60 timesteps.

5.1 Conclusion

The results suggest that the LSTM RNN was the worst performing model with the highest MSE and the shortest prediction accuracy. The ESN had the most accurate Lorenz attractor and the longest prediction accuracy, but the Transformers model had a lower MSE. Although the ESN model showed potential with less optimization effort, further research is needed to optimize and compare these models conclusively. The study indicates that the ESN is slightly better than the Transformers model, with the LSTM RNN performing the least effectively.

Future research could explore the NVAR reservoir computing variant and further optimize ESN hyperparameters. For the Transformers model, experimenting with different positional embeddings and self-attention mechanisms could be beneficial. This could help determine the full potential of both the ESN and Transformers models in predicting chaotic systems.

Appendix

A Predicted trajectories

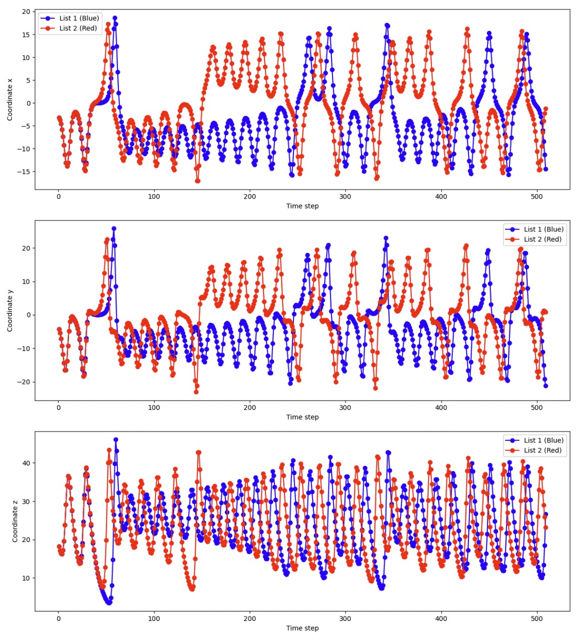


Figure 5: LSTM RNN path (true path is red, model path is blue)

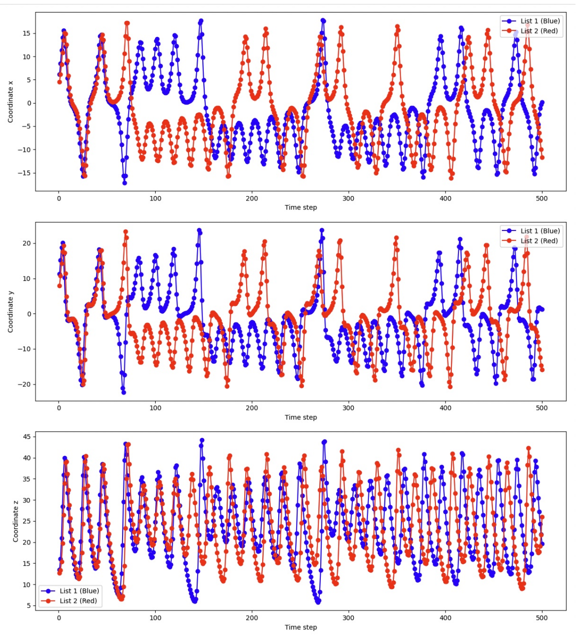


Figure 6: ESN path(true path is red, model path is blue)

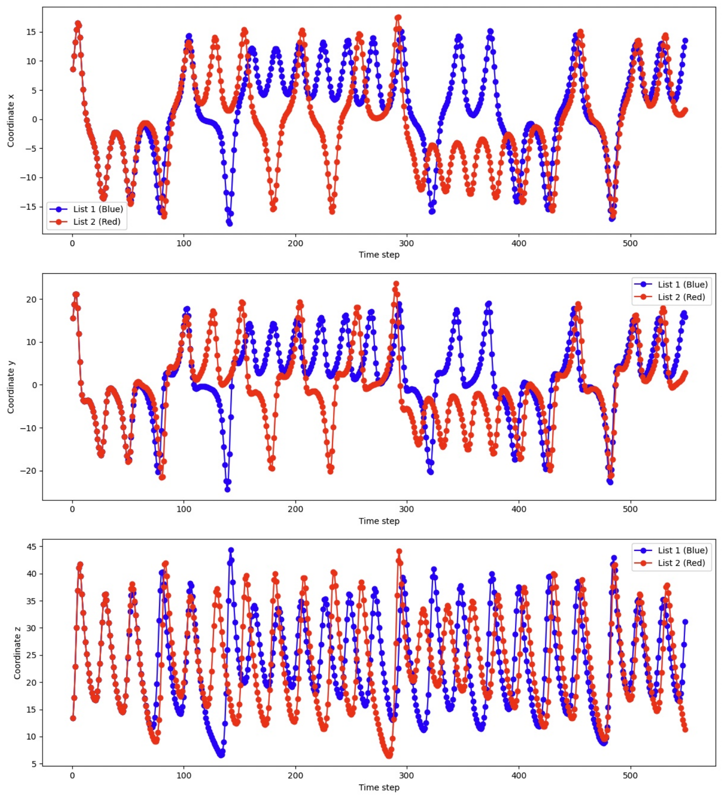


Figure 7: Transformers path(true path is red, model path is blue)