Forecasting Chaos with more Recurrent Neural Networks Architectures

Timur Zhanabaev

MAST 680 - Project - April 24, 2023

Abstract

In this work, we compare the performance of three RNNs; GRU, LSTM, and ESN in modeling a dynamic system, with the training generated from the chaotic Rossler system. The results show that all three models perform well in predicting short-term dynamics (one-time step), with quick divergence for long-term predictions. It was shown that LSTM with 100 unit outperforms the GRU and ESN models achieving maximal Lyapunov time of 4.79s. Although, more fine-tuning was required for the ESN, being sensitive to the choice of hyper-parameters.

1 Introduction and Overview

Recurrent neural networks (RNNs) are a powerful tool for modeling dynamic systems, which are systems that change over time. In particular, gated RNNs such as Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) networks, have been shown to be effective in modeling time-series data, as well as alleviating the vanishing/exploding gradient problem. In particular, GRU and LSTM have had great success in language model, where grammar follows a sequential structure.

Another type of RNN that has gained popularity for modeling dynamic systems is the Echo State Network (ESN). ESNs are a type of reservoir computing approach, which uses a fixed random matrix to embed the input to a higher dimension, capturing the complex non-linear dynamics of a system.

Searching for the best model, the three models are trained on a chaotic system, the Rossler system, and their performance is evaluated using the Lyapunov time as the forecasting horizon metric. Finally, a hyper-parameter search is conducted on all three models to determine the optimal RNN for accurately modeling the system.

2 Theoretical Background

2.1 Recurrent Neural Networks

Diverging from the architecture introduced in the course, where the recurrent layers take input of last output in s steps (compositions of network $g(\cdot)$), such model learns the best fit from point x_n to points $x_{n+1}, x_{n+2}, ..., x_{n+s}$, with the trainable parameters being W_j for each hidden layer.

In some more traditional recurrent networks, in addition to the hidden layers with trainable parameters W_j, the model has **state layers** with trainable parameters called *context units*. These context units store information of the previous state in are fed as additional input to fit the next time step, effectively the context units incorporate memory of sequential data.

2.2 The Jordan and Elman RNNs

To begin with the increasingly complex RNN architectures, the two simplest recurrent networks, Jordan and Elman RNNs, are presented as an introductions.

Let $x_t \in \mathbb{R}^p$ be our input, or another hidden layer's units. Then, $h_t \in \mathbb{R}^p$ is the hidden state at time t of the hidden units.

The Elman network recurrent layer is,

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

In the Elman layer, the hidden states are fed as input to generate the output of next time step $y_t = x_{t+1}$. In the recurrent fashion, the context unit h_{t-1} is the used again to encode the next h_t given input x_t and h_{t-1}.

In the Jordan's network, instead of feeding the hidden state information, the output of the network y_t is fed to encode the next hidden state h_t,

$$h_t = \sigma_h(W_h x_t + U_h y_{t-1} + b_h)$$

$$y_t = \sigma_y(W_y h_t + b_y)$$

These architectures are represented with diagrams (1).

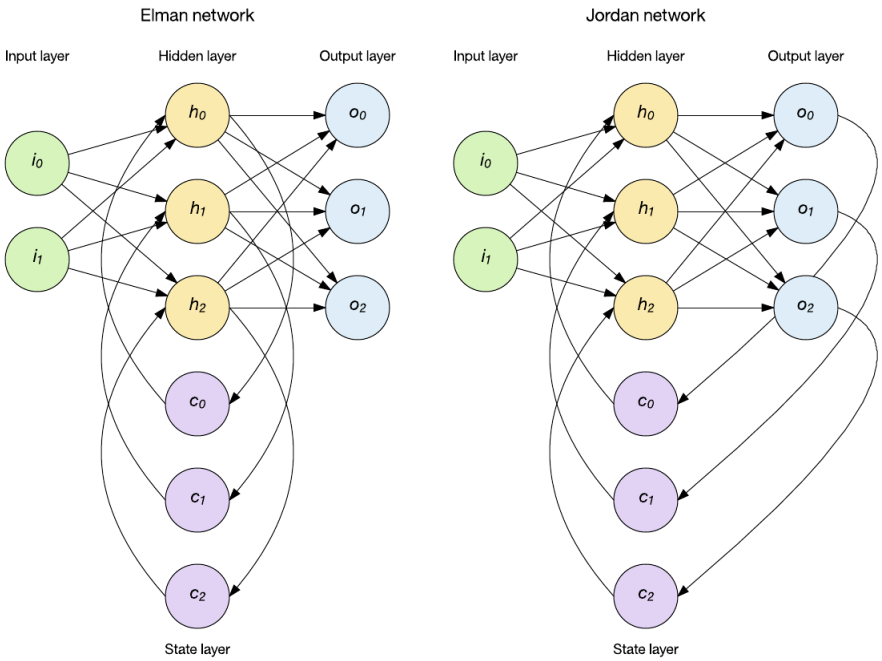

Figure 1: The Elman and Jordan network layer architectures [**?**]. (Here the hidden state layer with hidden state units is denoted as c_i)

2.3 The Vanishing Gradient Problem

A common problem arises for RNNs, where the gradients must be propagated through multiple time steps and layers, effectively reducing or increasing the gradient exponentially. This then leads to difficulties in model learning, where the gradient can either vanish, stopping the model from learning, or have the gradient explode preventing the model from converging to a solution [2].

2.4 The Gated Recurrent Unit Network

To circumvent the vanishing or exploding gradient problems, the Gated Recurrent Unit networks were designed to alleviate this issue by modulating the amount of past history to keep (modulating the passage of information through layers). These GRU layers, initially developed to simplify the LSTM architecture, extend the simple RNN layer to include *reset* gates that modulate weight of previous hidden states and the *update* gates modulate between newly generated *candidate states* or previous hidden state [3].

The reset gating mechanism is, where $r_t \in \mathbb{R}^{N_r}$ (N_r the number of reset units),

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

and update gates are then,

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

The *candidate state* is then a combination of input and the previous hidden state modulated by the reset weights $[r_t]_i$, defined as,

$$\hat{h}_{t-1} = tanh(W x_t + U(r_t \odot h_{t-1})$$

While the final state is then updated as a combination of candidate state and previous state,

$$h_t = (1 - z_t)h_{t-1} + z_t \hat{h}_{t-1}$$

With $[r_t]_i \approx 0$ the previous hidden states are forgotten. Hence input dominates in $\hat{h}_{t-1}$. As well with $[z_t]_i \approx 0$ we use the past hidden state to generate new hidden state, allowing the network to adaptively what to remember and forget.

2.5 The Long-Short Term Memory Network

Invented before the GRU Networks, the LTSM Networks were the first to be designed for the vanishing gradient problem by controlling the error flow through the model [4]. The LTSM has a more convoluted architecture with multiple gates, input, forget and output gates. Additionally, the layer has two memory states; the cell and hidden states.

With the input being again the p hidden layer units, then there is p gate units for each gate to match the dimensions. The gates and update functions are listed as follows,

the forget gate,

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

the input gate,

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

the output gate,

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

the cell input,

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

the cell state, with $\odot$ being the element wise multiplication (Hadamard product),

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

the hidden states,

$$h_t = o_t \odot \sigma_h(c_t)$$

Similarly to reset gate, the forget gates modulate the hidden state updates. While the input/output gates protect the memory contents [4]. These two architectures, GRU and LSTM are represented with diagrams, (2).

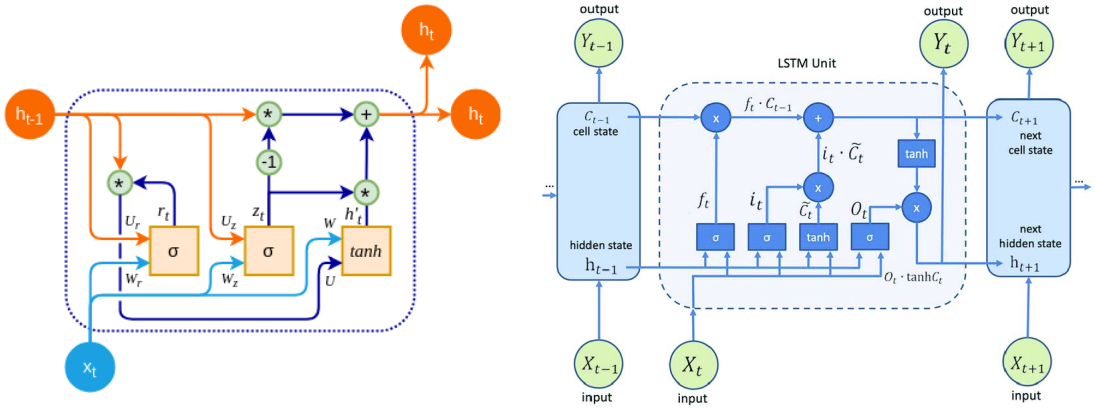

Figure 2: The GRU architecture [?] (left). The LSTM Architecture. [?] (right)

2.6 Echo State Network Architecture

For the final RNN architecture, we explore the Echo State Networks (ESN) which are also from the branch of *Reservoir Computing*, a paradigm where the input is mapped to a higher dimensional space (*an embedding*) having good success at modelling non-linear dynamics [2]. In the model, the *input weights* are fixed and sparsely connected to the ESN hidden units (called the *activation states* in the literature). For such model to be considered an Echo State Network, the model must then meet the *echo state property* requirements which are presented shortly.

Following the framework of [1], with some changes to notation to suit previous sections, the echo state network layer has an **input layer** of K units, an **internal layer** of N units (called the *reservoir*) and an **output layer** of L units. The input at time t is denoted as $x_t \in \mathbb{R}^K = (x_1^{(t)}, x_2^{(t)}, ..., x_k^{(t)})$, that is our dynamic system. The internal units encoding is $h_t \in \mathbb{R}^N = (h_1, ..., h_n)$ (activation of internal units) and are updated as,

$$h_{t+1} = f(W^{\text{in}} x_{t+1} + W^{\text{res}} h_t + W^{\text{back}} y_t)$$

where,

$$W^{\text{in}} \in \mathbb{R}^{(N \times K)}, W \in \mathbb{R}^{(N \times N)}, W^{\text{back}} \in \mathbb{R}^{(N \times L)}$$

and f is typically a sigmoid function and the output units are denoted as y_t (the feedback output, not the next time iterates of x_{t+1}), which are updated as well by following function,

$$y_{t+1} = f^{\text{out}}(W^{\text{out}}\langle h_{t+1}, x_{t+1}, y_t\rangle)$$

where $f^{\text{out}}(x) = x$ is the identity activation function for our dynamic system, with trainable parameters $W^{\text{out}} \in \mathbb{R}^{(L\times(K+N+L))}$, acting upon the concatenated vectors h_{t+1}, x_{t+1} and y_t. The graph representation of the network is displayed in Figure (3), where N internal units represent the reservoir, with edges connecting to the reservoir from K input and L output units.

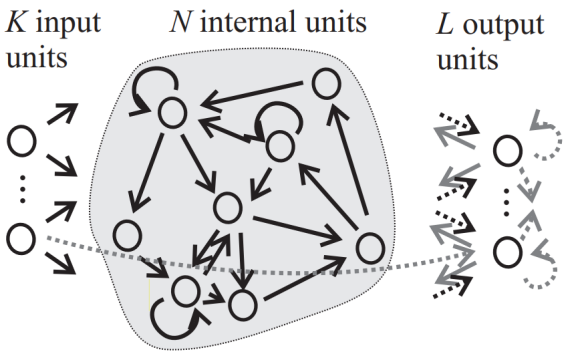

Figure 3: The basic ESN architecture. The arrows representing the weights and nodes the layer units. [1]

2.7 Admissible States

The input is required to be taken from the dynamic system's *admissible states*, that is the space of the system. As well as the network states lie in a compact set. These conditions are called *standard compactness conditions* [1].

2.8 Echo State Property

Among some conditions that the network must hold to give the RNN hidden units echo state properties, but not easily checked in practice; *contracting network, state forgetting network* and *input forgetting network* [1]. The practical sufficient condition to obtain echo states require the weight matrix W^{res} maximum singular value to be less than 1,

$$\sigma^{\max}(W^{\text{res}}) < 1$$

Then the matrix $\tilde{W}$ is then scaled by $\alpha_{\min} = 1/\sigma_{\max}(\tilde{W})$ to get $W = \alpha_{\min}\tilde{W}$. As well the matrix can be scaled by the spectral radius of $\tilde{W}$, $\alpha_{\max} = 1/|\lambda_{\max}|(\tilde{W})$ to get $W = \alpha_{\max}\tilde{W}$. Now having a range of values $\alpha \in [\alpha_{\min}, \alpha_{\max}]$, choosing a lower α gives a greater probability of echo states [1]. Although in the end, the echo states can happen even if condition is not met, as it depends highly on the *admissible states* [1]. Moreover with $|\lambda|$ being close to 1, "the slower is the decay of the network's response to an impulse input" [1], recommended to model periodic signals.

A common procedure to generate $\tilde{W}^{res}$ matrix is to initialize it as a sparse matrix, with weights usually being {-0.4,0,0.4} with probabilities of {0.025,0.90,0.025}, a sparse connectivity of 5% as an example to hold the echo state property.

Thus reservoir matrix W^{res} is kept static in the ESN, where the weights W^{in} and W^{out} are trainable.

2.9 Leaky Integrator Neurons

For learning slowly and continuously changing systems, it is more adequate to use networks with a continuous dynamics. We take a hybrid approach here and use discrete-time networks whose dynamics is a coarse approximation to continuous networks with leaky integrator units (LIU). The activation unit then evolves according to the dynamics, as a difference equation with step size δ [1],

$$h_{t+1} = (1 - \delta C\gamma)h_t + \delta C \left\{ f(W^{\text{in}} x_{t+1} + W^{\text{res}} h_t + W^{\text{back}} y_t) \right\}$$

where C is a *time constant* and γ is the *leaking decay rate*. The condition for existence of echo-states requires the value to be $|1 - \delta C(\gamma - \sigma^{\max}(W))| < 1$.

2.10 The Rössler System

For the subject of this study, the Rösslerdynamic system is generated as the sequential data for the RNN models, trained to predict the future steps of the chaotic system. The differential equations that define the system are,

$$\dot{x} = -y - z$$

$$\dot{y} = -x + ay$$

$$\dot{z} = b + z(x - c)$$

The parameters $a = 0.1$, $b = 0.1$, $c = 18$ show a stable chaotic attractor in Fig (4).

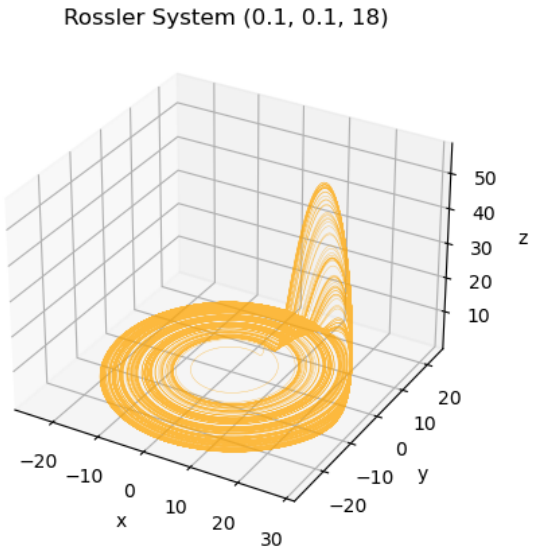

Figure 4: The Rössler System - A Chaotic Attractor; parameters a = 0.1, b=0.1, c=18.

3 Algorithm Implementation and Development

3.1 Simulation

Using the `scipy`'s ODE solver, the Rössler system is simulated for the parameters; $a = 0.1$, $b = 0.1$ and $c = 18$, rendering the system chaotic. The goal is then to simulate data from multiple initial conditions to fully cover the trainable space (admissible states) of the system. For the continuous-time system, at a time increment of $\triangle t = 0.01$, 50000 points are generated for the training data, resulting in an interval $t \in [0, 500]$, starting from the initial condition $(5, 5, 5)$.

3.2 Training Data

The RNN input is of shape (`batches`, `n_samples`, `features`), where `batches` is the number of observations (sequences of time length `n_samples`). Then `n_samples` is seen as number of time points in the sequence, i.e. $\{x_1, ..., x_{n\_samples}\}$ used to predict the next state $x_{n\_samples+1}$. Finally, `features` is the dimension of the system, the three spatial coordinates (x, y, z) in our case. For 50000 data points, the training dataset is then transformed to $(49989, 10, 3)$, using 10 steps to predict 1 step into the future (leaving out 11 points from the windowing of each sequence).

3.3 Test Data

After the training, the test dataset is transformed the same way. Starting from the initial condition $(1.39, 19.46, 0.019)$, being the last state of the training data, to allow the model then continue the dynamics for a sequence of 5000 time-steps at a 0.01s time-increment. This gives us then 4989 sequences of dimension $(10, 3)$ for testing.

3.4 RNN Implementations

For the RNN implementation, the choice of language is Python with the `TensorFlow 2` Machine Learning framework. The networks are then sequential models, with first layer being fully connected `Dense(units)` layer, then a Recurrent Layer; `GRU` or `LSTM` or `ESN`, finishing with the output layer `Dense`(3). All three models use the `Huber` loss function with the `Adam` optimizer.

3.5 RNN Forecasting

To assess the performance of each model, the models are made to predict the future time step from unseen data (test data), with the root squared error (*euclidean distance*) plotted across the time span. Hence for the RNNs, the model can be fed the test data in the form of (`batches`, `n_samples`, `features`) to predict each future time step from simulated data, as shown by eq. (1), where g is a network.

$$x_{t+10}^{(\text{pred})} = g(\{x_t^{(\text{true})}, ..., x_{t+9}^{(\text{true})}\}) \tag{1}$$

The second way is to allow the model **warm-up**, that is to predict the first 10 outcomes and then use the predictions to as input to predicting the next steps, allowing the model to diverge and effectively measuring their performance with Lyapunov time, where the true data diverges by $e \approx 2.71$ from the predicted data, with the difference shown in eq. (2),

$$x_{t+10}^{(\text{pred})} = g(\{x_t^{(\text{pred})}, ..., x_{t+9}^{(\text{pred})}\}) \tag{2}$$

For the ESN network, the parameters include; the leaky rate, connectivity and units number. For GRU and LSTM networks, the parameters include; units number and activation functions. The results for tested parameters are listed in table (1).

4 Computational Results

4.1 One-Step Prediction

Training the initial 3 default models, GRU with 50 units for 150 epochs, LSTM with 50 units for 150 epochs and ESN with 50 units for 150 epochs, `connectivity` = %5, `leaky` = 0.1 and `spectral_radius` = 0.8. As well for ESN (only activation function), GRU and LSTM the default **activation** and **recurrent activation** functions are **tanh** and **sigmoid**, respectively. The forecasting method type 1 is used to predict the future step for each batch, as shown in Fig (5).

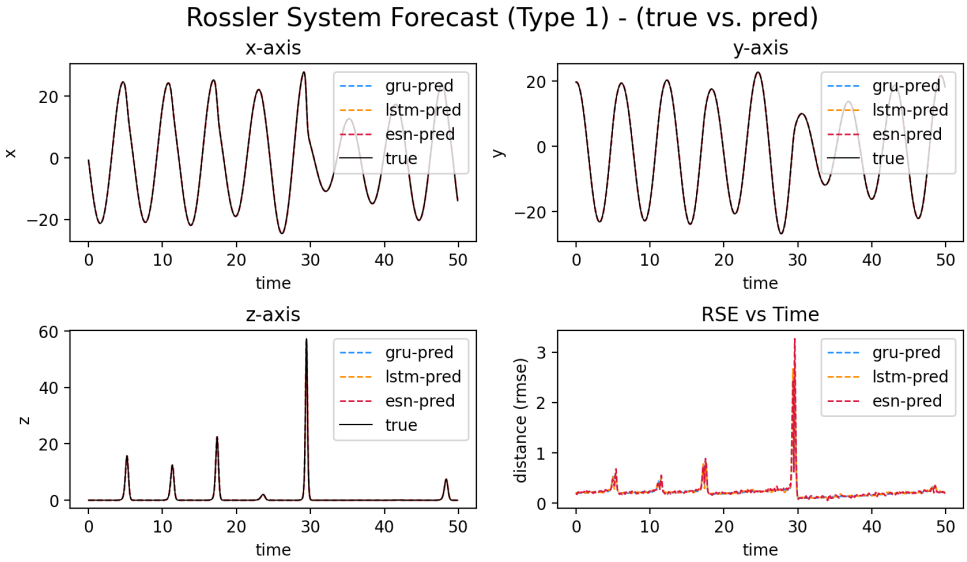

Figure 5: Rossler System (0.1, 0.1, 18) Forecast (type 1): The 3 models (GRU in blue, LSTM in orange, ESN in red) are plotted for each axis, with the time-series RSE plotted (bottom right).

The 3 models are able to then predict the next time step with good precision, MSE remaining in $[0, 2.5]$ across time, with greatest deviations when they traverse the *hump* of the system.

4.2 N-Step Prediction

Although, the last section is not an indicator of the model being predictive over a large horizon. To assess the performance over an undetermined amount of time, the models are let free to predict from the **warm-up** stage until e divergence from true data, as shown for **LSTM model of 100 units, sigmoid recurrent activation and tanh activation**, having achieved the best results out of all models and replicating the *the spirit* of the system's dynamics, with Lyapunov time of 4.23s, in Fig. (6).

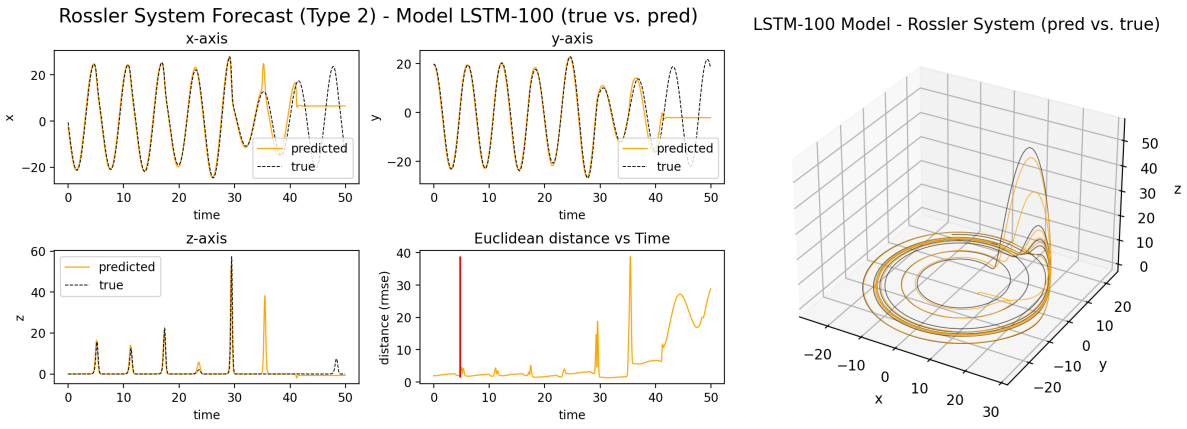

Figure 6: Rossler System (0.1, 0.1, 18) Forecast (type 2): The LSTM Model with 50 units is plotted for each axis (true vs. predicted data) and RSE over time, with Lyapunov time marked at $t = 4.79$s. The right figure plots the forecasted dynamics vs the true dynamics.

For the 3 *default* models of 50 units each; GRU, LSTM and ESN the Lyapunov times were measured to be 1.35s, 1.7s and 2.02s. Showing promising results, when comparing to the theoretical values of chaotic systems' Lyapunov times being in the same ranges for the chosen dynamic system parameters. The default methods are listed in the appendix in Fig. (7), (8) and (9).

Performing the same experiments on more parameters (number of units), tables are built for each model and their measured Lyapunov times are shown in table (1), with a larger number of hidden units generating longer forecasting horizon.

hidden units	10	25	50	100
GRU L-time	0.17	0.95	1.35	1.71
LSTM L-time	0.11	2.68	1.7	4.79
ESN L-time	0.04	0.3	2.02	1.65

Table 1: **Model Tuning**: GRU, LSTM, ESN (L-time being Lyapunov time). default functions: `activation` = `tanh`, `recurrent_activation` = `sigmoid`.

4.3 Tuning the ESN Hyper-parameters

A 200 unit ESN is trained for varying parameters; `leaky`, `connectivity`, `spectral_radius`, where the metric in each cell is the Lyapunov time, shown in table (2). The results show that the ESN is sensitive to the leaky rate choice with ideal choice being 0.1 for the system at a 0.01 time increment/ As well, the spectral radius closer to 1 tended to forecast the data better over longer range. In this case, the sparsity has been kept low 5% and 10%, without deviating from the intended design of echo state networks, even though now difference was observed between the two values.

Lyapunov Time Measurements by Hyper-parameters

sparsity 0.05

	leaky 0.01	**leaky** 0.1	**leaky** 0.5
spectral radius 0.5	0.21	2.17s	0.56s
spectral radius 0.8	0.12	1.69s	0.67s
spectral radius 0.95	0.22	2.23	0.58s

sparsity 0.1

	leaky 0.01	**leaky** 0.1	**leaky** 0.5
spectral radius 0.5	0.17	1.64s	0.39s
spectral radius 0.8	0.23	2.98s	0.99s
spectral radius 0.95	0.17	2.12s	0.2s

Table 2: Hyper-parameter tuning for ESN; leaky rate, connectivity and sparsity parameters. (L-time being Lyapunov time).

5 Summary and Conclusions

In conclusion, it was shown that RNNs are able to learn the system's dynamics for very short-term predictions, and in some cases are able to forecast the systems at reasonable ranges based on the theoretical Lyapunov time of the system. Specifically, GRU and ESN tended to predict shorter ranges, while LSTM excelled at longer ranges. It was shown as well that while greater hidden unit size increased performance, the **100 hidden-units LSTM** was able to stabilize itself to continuously predict a chaotic trajectory of the system, while other models tended break down once they passed the Lyapunov time. The possible reason that models breakdown is due to lack of training data outside the basic of attraction where the RNNs diverge into. The possible way to circumvent this is then to train the network on all possible initial states to assure that the model returns to the basic of attraction, which would be computationally expensive.

References

[1] Herbert Jaeger. *The "echo state" approach to analysing and training recurrent neural networks, Fraunhofer Institute for Autonomous Intelligent Systems*, January 26 2010.

[2] Shahi S, Fenton FH, Cherry EM. *Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study.* Mach Learn Appl. 2022 Jun 15.

[3] Kyunghyun Cho, et al. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation.* In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP).

[4] Hochreiter, Sepp & Schmidhuber, Jürgen. *Long Short-term Memory.* Neural computation (1997)

Appendix A.1 Premade functions used and brief implementation explanation.

1. `plot(plot(pred_data, true_data, model_name)`: takes in forecasted data and true data to plot the system's dynamics over 3 axes and the RSE over time.
2. `forecast_t1_plot(models, test_input, test_target)`: Forecasting Type 1. Uses the one step forecasting using a model for all test batches (test_input). The plots are then made comparing to the test output (test_target) for all axes and the RSE over time.
3. `forecast_t2(model, test_input, in_steps, samples)`: Forecasting Type 2. Uses a model to simulate the dynamics from a warm-up determined by in_steps for n iterations.
4. `Rossler(x, t, a, b, c)`: the system's dynamics input function to the `odeint` solver.
5. `GRUCell(units, activation, recurrent_activation)`: The GRU layer fed as input to the RNN layer.
6. `LSTMCell(units, activation, recurrent_activation)`: The LSTM layer fed as input to the RNN layer.
7. `ESN(units, connectivity, leaky, spectral_radius)`: The ESN layer from tensorflow addons package.

Appendix A.2 Extra Figures

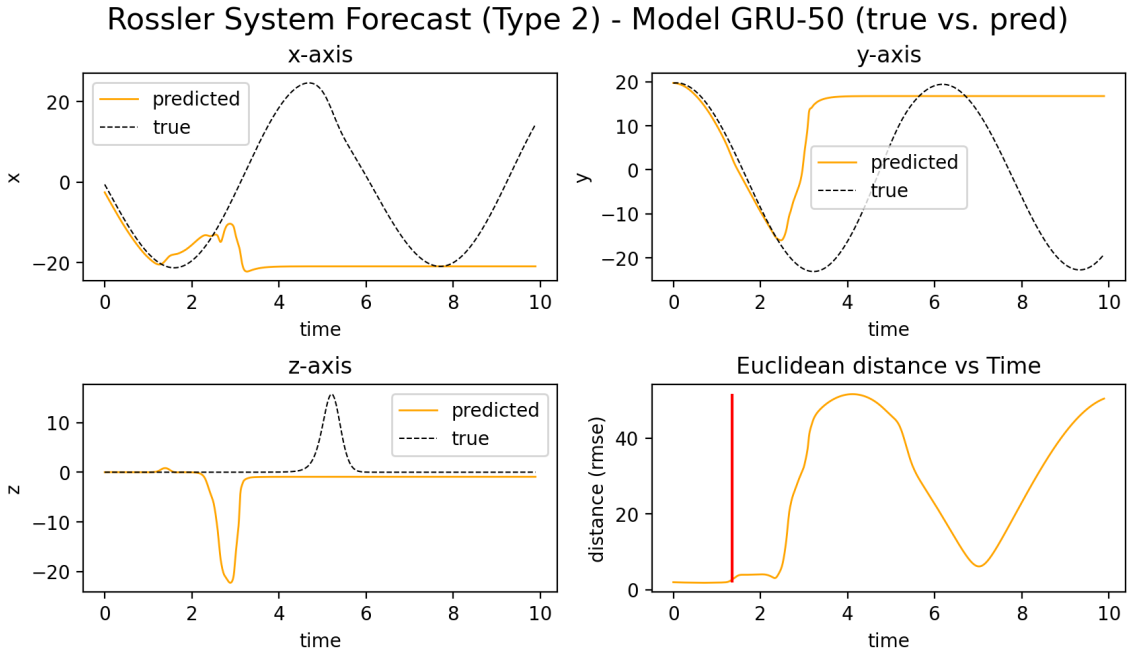

Figure 7: Rossler System (0.1, 0.1, 18) Forecast (type 2): The **GRU-50** model is plotted for each axis (true vs. predicted data), with the time series RSE plotted (bottom right).

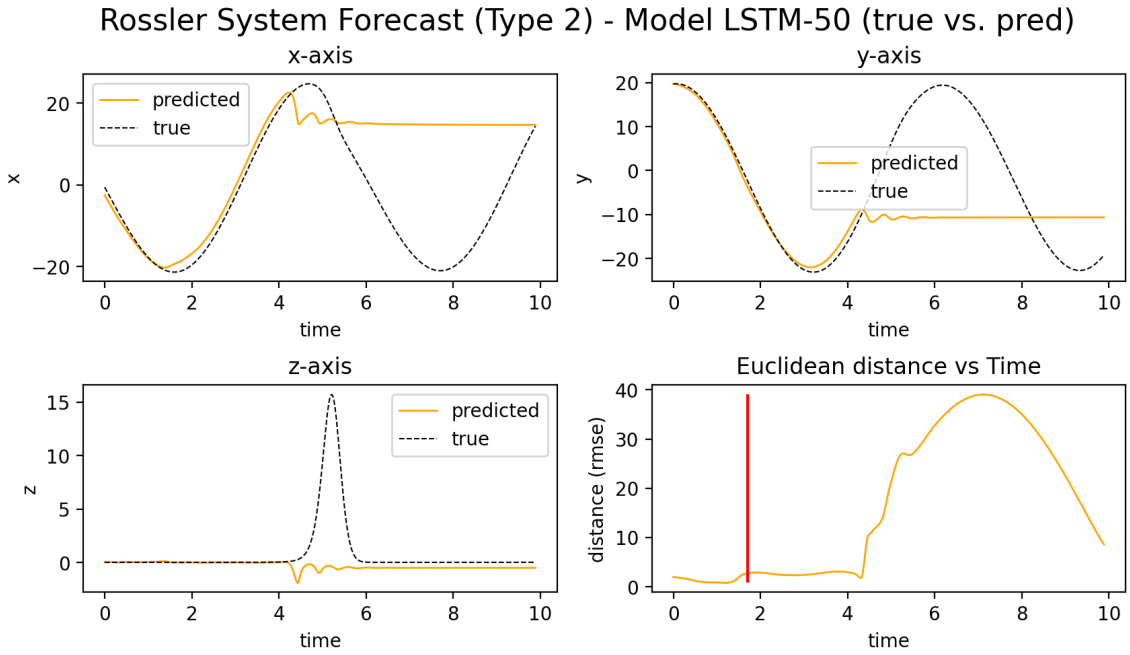

Figure 8: Rossler System (0.1, 0.1, 18) Forecast (type 2): The **LSTM-50** model is plotted for each axis (true vs. predicted data), with the time series RSE plotted (bottom right).

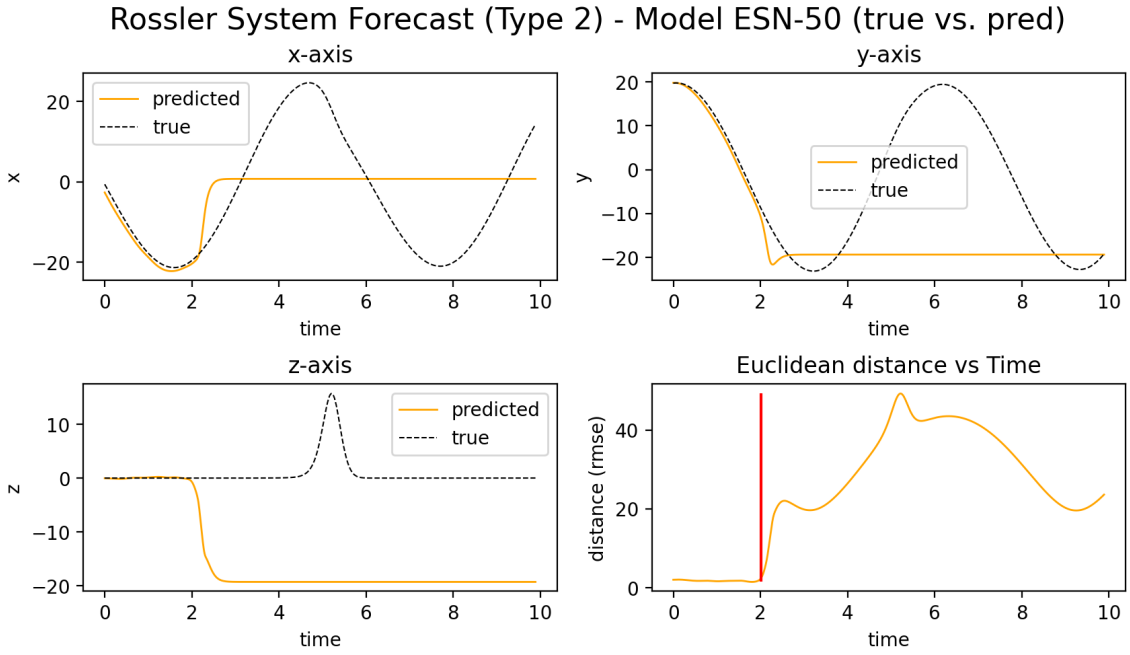

Figure 9: Rossler System (0.1, 0.1, 18) Forecast (type 2): The **ESN-50** model is plotted for each axis (true vs. predicted data), with the time series RSE plotted (bottom right).

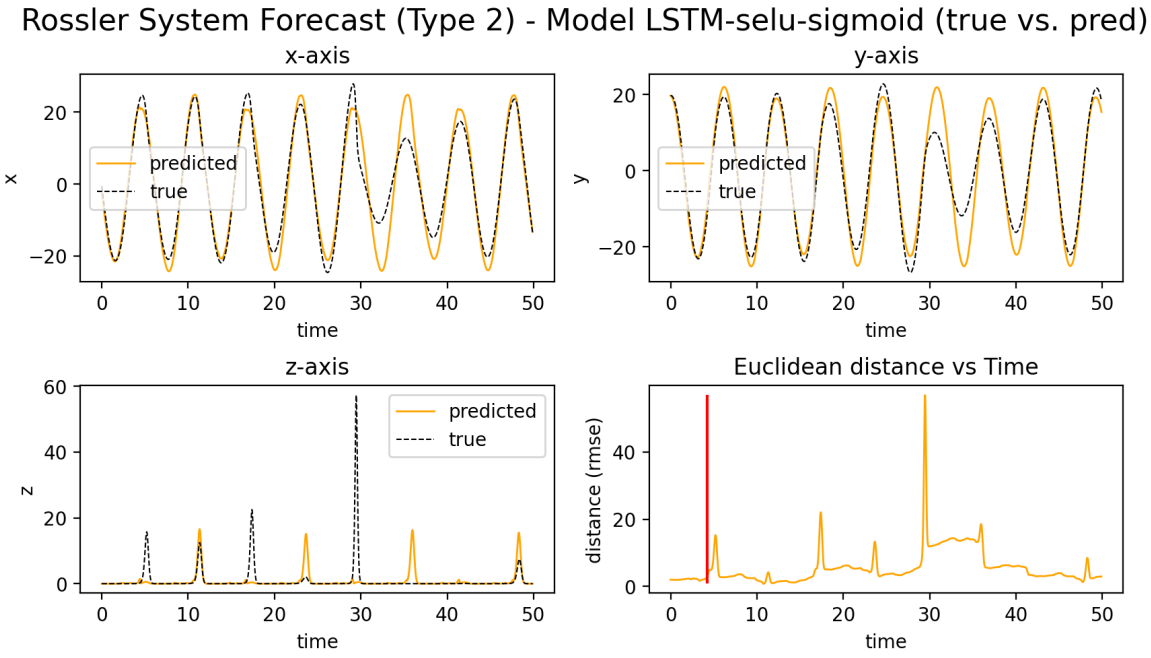

Figure 10: Rossler System (0.1, 0.1, 18) Forecast (type 2): The **LSTM-100** (activation=**selu**, recurrent_activation= **sigmoid**) model is plotted for each axis (true vs. predicted data), with the time series RSE plotted (bottom right).

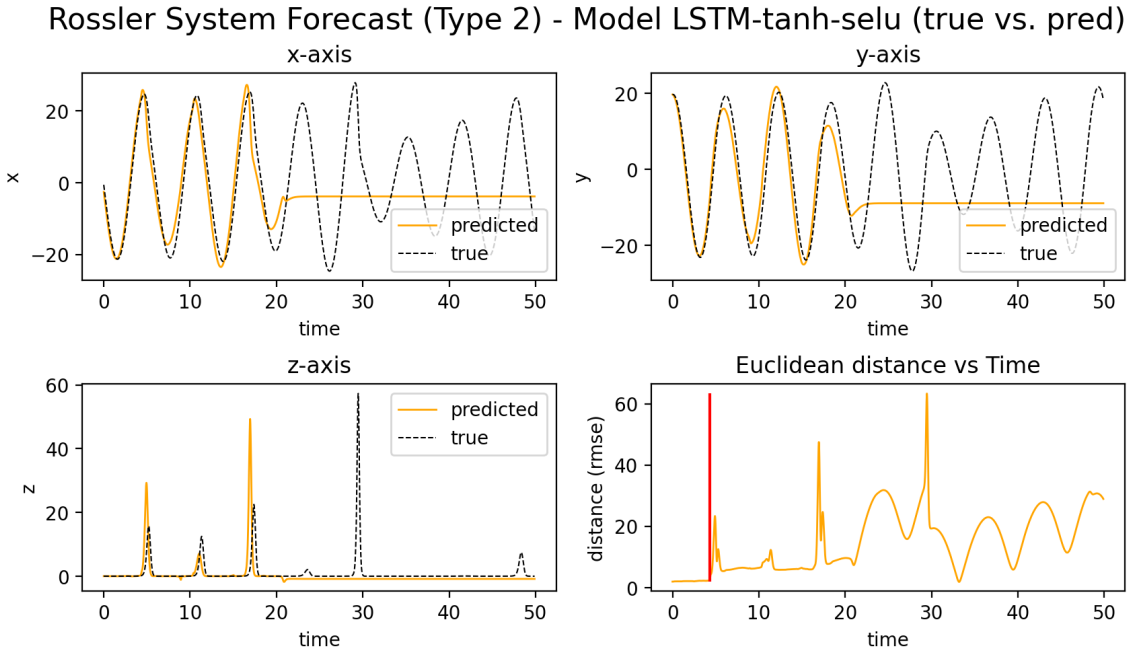

Figure 11: Rossler System (0.1, 0.1, 18) Forecast (type 2): The **LSTM-100** (activation=**tanh**, recurrent_activation=**selu**) model is plotted for each axis (true vs. predicted data), with the time series RSE plotted (bottom right).