An analysis of the LSTM's ability to learn and reproduce Chaotic Systems.

Sheil Kumar
University of Illinois Urbana-Champaign
sk17@illinois.edu

***Abstract*— a-**

I. Introduction

The constantly evolving and diversely applicable field of machine learning has grown exponentially as both the magnitude and accessibility of computing power have improved in recent times. Accompanying this growth, great strides have been made in testing the limits of the various machine learning frameworks. The frameworks have also been used to try and understand chaotic systems, while previous methodologies would suggest a knowledge based approach to model the equations governing these systems, machine learning methods aim to predict future states of the system by relying just on past time series data, providing a model-free method of accurately predicting these systems.

The Long-Short Term Memory (universally abbreviated LSTM) cell is a hidden unit belonging to the family of artificial Recurrent Neural Networks (RNNs) which was designed to be capable of remembering long term dependencies[1] which previous RNNs found difficult to address due to the vanishing or exploding gradients problem that occurs while back-propagating in time during the training of said RNNs. Networks utilizing LSTMs have already demonstrated their capacity to solve problems with long term dependencies as they have been used in analysis of audio[2] and video[3] data, speech recognition[4], and predictions of traffic flow[5].

There have been many attempts at simulating and reproducing the dynamics of the 1963 Lorenz system[6]. Using machine learning schemes to reproduce the Lorenz system is not a novel task in the field, as various machine learning frameworks have been used to tackle this problem, attempts using Reservoir Computing schemes[7–10] have been thoroughly explored with varying degrees of success, especially in the short term. Even the use of LSTMs in modelling the Lorenz has been explored to some extent, [11] explores the effects of precision of the training data in predicting future time steps of the Lorenz system compared to the true values. In this paper we explore the effect of the LSTMs hyper-parameters in predicting the dynamics of the system, we demonstrate each designed networks ability to predict the system by evaluating the errors between the predicted and true values of the attractor, as well as how well the dynamics are modelled by comparing the maximal Lyapunov exponents of a predicted system to that of the literature value.

II. Long Short Term Memory (LSTM)

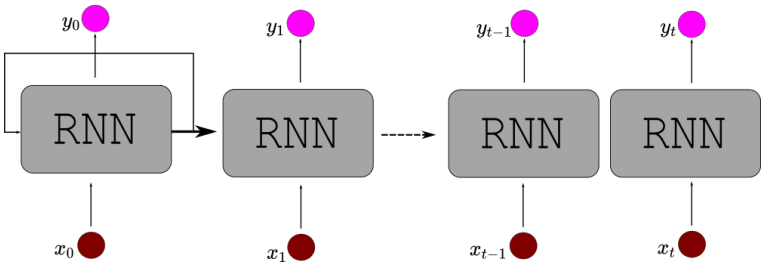
The LSTM belongs to the family of RNN cells, it was designed in order to address the vanishing/exploding gradients problem that plagues traditional RNN cells. The structure of a regular RNN layer and the schematics of an LSTM cell can be seen in Fig. 1 and it's governing equations are provided in Table I.

TABLE I: LSTM Equations

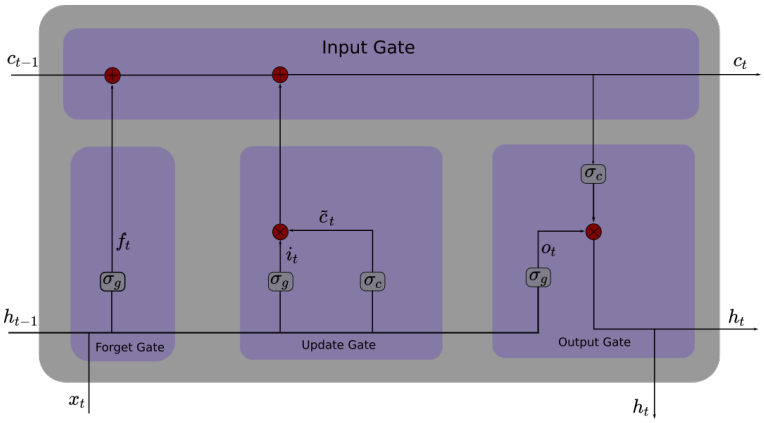
Forward Pass Equations
$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$
$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$
$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$
$\widetilde{c_t} = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$
$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t}$
$h_t = o_t \circ \sigma_c(c_t)$
Notation
$x_t \in \mathbb{R}^d$: input vector to LSTM cell
$f_t \in \mathbb{R}^h$: forget gate vector
$i_t \in \mathbb{R}^h$: input gate vector
$o_t \in \mathbb{R}^h$: output gate vector
$h_t \in \mathbb{R}^h$: hidden state vector of LSTM cell
$\widetilde{c_t} \in \mathbb{R}^h$: cell input vector
$c_t \in \mathbb{R}^h$: cell state vector
$W \in \mathbb{R}^{h \times d}$: weight matrix for input vector[1]
$U \in \mathbb{R}^{h \times h}$: weight matrix for hidden state vector
$b \in \mathbb{R}^h$: bias vectors
Activation Functions
σ_g : sigmoid function
σ_c : hyperbolic tangent function

[1] the values d and h refer to the number of input features (in our case 3 (x,y,z) components of the time step) and the number of hidden units respectively.

A LSTM layer is a collection of h horizontally stacked LSTM cells that feed information back to one another. Each LSTM cell itself comprises a network of 4 gates, the forget



(a) Default RNN Layer



(b) LSTM Cell Schematic

Fig. 1: Schematic of a LSTM Cell and the structure of a normal RNN layer.

gate, input gate, update gate, and the output gate. These 4 gates oversee the operation of functions which resemble the mechanisms of 4 separate dense layers, each containing two independent weight matrices and a bias vector: W_f, W_i, W_c, and W_o which act on the input vector x_t; U_f, U_i, U_c, and U_o which act on the previous hidden state vector h_{t-1}; and bias vectors b_f, b_i, b_c, and b_o. Each of these matrices and vectors belong to their respective gates and have dimensions as detailed in Table I. Thus for any LSTM layer, we have a total of 12 parameter holding matrices and vectors and the total number of parameters in any LSTM layer is $4(hd+h^2+h)$ where h is the number of LSTM cells/units in the layer and d is the number of features responsible for describing any time step of the data.

During any forward pass of the LSTM layer, each LSTM cell is provided with 3 inputs, x_t and h_{t-1} as mentioned above, as well as the previous cell state vector c_{t-1}. If a previous hidden state and previous cell state vector do not exist, they are initialized with zeros. The input vector and the previous hidden state vector are passed to all the gates where they are acted upon by their respective weights and biases and then activated by an activation function. The forget, input and output gate all use a sigmoid activation function, however the update gate uses a hyperbolic tangent activation function; the reasoning behind this is understood after exploring each gates purpose. The forget gate creates the forget gate's activation vector f_t and is responsible for deciding if the information is important to keep or not and is thus activated by the sigmoid function to scale the values between 0 and 1, with 1 being very important and 0 not being important. The forget vector is then point-wise multiplied by the previous cell state vector to see which information should be kept and which should be dropped. The update gate oversees the creation of both the input gate activation vector i_t as well as the cell input vector $\tilde{c}_t$ while the input gate vector is activated by the sigmoid function, the cell input vector is activated by the hyperbolic tangent function. In this manner, the sigmoid function decides which values need to be updated and the tanh function regulates the network, the two are then point-wise multiplied and thus the sigmoid output decides what is to be kept from the tanh output. At the input gate, we already have the point-wise multiplication of the forget vector and the previous cell state, this is then added to the return of the update gate, the point-wise product of i_t and $\tilde{c}_t$ to create the new cell state vector c_t which will be passed back to the LSTM cell during the next time step. Finally, the output gate sees the production of the output gate activation vector o_t and the new hidden state vector h_t. The output vector is activated once again by the sigmoid function, while the new cell state vector is passed to a tanh function, these two are then point-wise multiplied to create the new hidden state h_t which will also be passed back into the LSTM cell during the next time step. In this way, the sigmoid function decides which part of the current cell state vector should be kept by the hidden state vector. In summary, the forget gate decides what information should be kept from previous steps, the input and update gates decide what information from the current step is important, and the

output gate decides the next hidden state. Once again refer to Table I for the precise equations responsible for governing each gate and the creation of their respective vectors.

III. Data

A. Data Generation

We generate data from the Lorenz attractor defined by the equations:

$$\begin{aligned} \dot{x} &= \sigma(y-x) \\ \dot{y} &= x(\rho-z)-y \\ \dot{z} &= xy-\beta z \end{aligned} \tag{1}$$

Where $\sigma, \rho,$ and β are constants, these are set to their classical values of 10, 28 , and 8/3 respectively.To generate the data, we make use of the Forward Euler Integration Scheme which works iteratively by solving

$$y'(t) = f(t, y(t)) \tag{2}$$

$$\begin{aligned} y(t_0) &= y_0 \\ t_n &= t_0 + ns \\ \therefore y_{n+1} &= y_n + sf(t_n, y_n) \end{aligned} \tag{3}$$

Where s is the **step size** and t_i is the i^{th} **time step**, we set step size $s = 0.01$. Identical equations exist for both x and z as well. We begin with the initial value $(x_0, y_0, z_0) = (0,0,0)$ and solve iteratively as defined above to obtain $\mathbf{X} = X_0, X_1, ..., X_n$, where $X_i = (x_i, y_i, z_i)$ for $n = 19200$.

B. Training and Testing

Let us define the target vector, $\mathbf{Y} = Y_0, Y_1, ..Y_{n-1}$, where $Y_i = (x_{i+1} - x_i, y_{i+1} - y_i, z_{i+1} - z_i) = X_{i+1} - X_i$. The target vector $\mathbf{Y}$ is the vector we will **train the model to predict**. As such the model will be fed the true $\mathbf{X}$ and true $\mathbf{Y}$ and be asked to predict a $\hat{Y}_i$ for each time step. That is, each X_i will be mapped to a $\hat{Y}_i$ as predicted by the model; $X_i \longmapsto \hat{Y}_i$. We then set the model to predict the $\mathbf{Y}$ by trying to minimize the loss function $\mathbf{L} = (\mathbf{Y} - \hat{\mathbf{Y}})^{\mathbf{2}}$, this loss function is simply the mean squared error.

We now have the data used for training, the true $\mathbf{X}$, and the data we wish the network to predict, the target vector $\mathbf{Y}$. The true $\mathbf{X}$ are then arranged into 128 sequential vectors of length 150, thus, there are 128 vectors each containing 150 time steps of the system. These are then split into training and testing sets in a 80% to 20% split - 80% of the data is used for training 15360 time steps, and 20% of the data is used for testing 3840 time steps. So the training data passed to the network, will have shape $[102, 150, 3]$ thus the training data is the set $\mathbf{X}_{\text{train}} = [\mathbf{X_1}, \mathbf{X_2}, ..., \mathbf{X_{102}}] = [[X_1, ...X_{150}], [X_{151}, ..., X_{300}], ..., [X_{15151}, ..., X_{15300}]]$ where each $\mathbf{X_i}$ is a vector of 150 time steps $[X_i, ..., X_{i+150}]$ where each $X_i = (x_i, y_i, z_i)$ in the vector is simply a set of coordinates on the Lorentz system. Similarly passed along with the training data will be the desired target vectors which will have shape $[102, 1, 3]$, thus the target vectors are the set $\mathbf{Y_{train}} = [[Y_1], [Y_2], ..., [Y_{102}]]$, where each $Y_i = (x_{i+1} - x_i, y_{i+1} - y_i, z_{i+1} - z_i) = X_{i+1} - X_i$ since we wish to train the network to use the previous 150 time steps to predict the **difference** between the 151st and the 150th time steps.

As discussed in Section II we now begin training the network by passing both the training and testing sets into the network itself. The network then initializes a set of random matrices $\mathbf{W}$ and $\mathbf{U}$ as well as a set of bias vectors $\mathbf{b}$ which are updated during training using batch gradient descent. The number of times network updates the parameters then depends on the batch size itself, in all the different networks we designed (by changing the number of LSTM units and changing the number of epochs), the batch size was kept at 2. So the network training and paramater updates were as follows, for each of epoch of training the network is fed the entire 102 consecutive sets of 150 time steps of the true $\mathbf{X}$ and true $\mathbf{Y}$ in batches of 2 consecutive steps of 150 time steps. The network then subjects these 2 sets to the forward pass equations as described in Table I, and discussed in Section II, calculates the loss using the loss function $\mathbf{L}$, and then begins back-propagating through time. During back-propagation, gradients for the different weights are calculated by differentiating the loss function with respected to the various different weights and biases, these are then used to update the various weights and biases after which the next batch of 2 is passed into the network for forward pass. This algorithm continues until all the batches have been through the network after which one epoch is concluded, and this continues for each epoch thereafter. Since the original shape of data fed into the network was $[102, 150, 3]$ and the chosen batch size is 2, during each epoch of training, the parameters $\mathbf{W}, \mathbf{U}$, and $\mathbf{b}$ are updated 51 times. These weights are then stored so that they can be imported for prediction.

C. Prediction

Our goal in creating these networks is two-fold, we wish to satisfy two requirements:

1) Dynamics: - The trained network should closely simulate the dynamics of the original system; it should have a Lyapunov exponent that is extremely close to the literature value corresponding to the system with $\sigma, \rho,$ and β as described in III-A.

2) Prediction Accuracy: - When asked to make consecutive predictions after being exposed to a trajectory on the attractor, the network should be able to accurately predict how the trajectory continues for a period of time without any further guidance or corrections.

While III-C.1 and III-C.2 may sound similar, there is one distinct difference, while *Prediction Accuracy* is as straight forward as it sounds, in modelling the *Dynamics* of the system we also wish that when the system is exposed to a minuscule perturbation, which grow exponentially in chaotic systems such as the Lorentz, the network is able to simulate

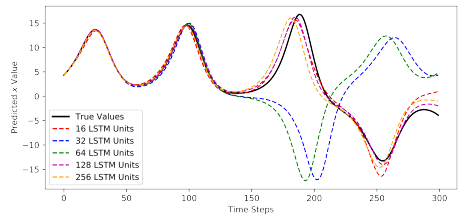
the dynamics of the system and pick up a new trajectory that is still on the attractor at a different time.

To meet these goals, we create a new network with a designated batch size of 1 and import the weights stored for its associated number of LSTM units. To satisfy III-C.2 this network is then exposed to a previously unseen sequence of 1000 consecutive time steps, $\mathbf{X}_{\text{train}} = [X_j, ..X_{j+1000}]$, to adjust it's internal state after which it is asked to predict X_{j+1001}, this value is then stored and fed back into the network and the network is asked to predict X_{j+1002} and this continues until X_{j+1300}. The results can be seen in Fig. 2, and will be discussed in section V. To meet the demands of III-C.1 we ...

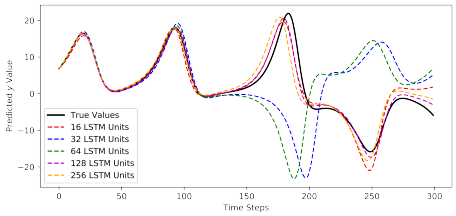
IV. Model

A. Network Structure

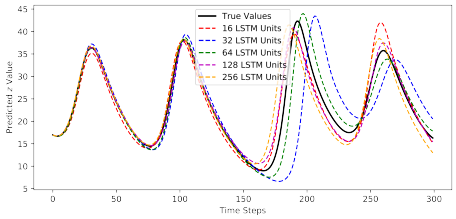
We have analyzed the Lorenz attractor through many different neural networks. However, the structure of the neural networks remains the same throughout and they all received the same data as described in **2.1**. Each neural network is first composed of an LSTM layer composed of a certain number of LSTM units, this is then fed into a dense layer composed of three units, respectively the (x, y, z) components of the attractor, that is then returned to the user by the network. We evaluated networks LSTM units ranging between 8 and 256 units, while the number of epochs were also varied.



(a) X-Coordinate Predictions



(b) Y-Coordinate Predictions



(c) Z-Coordinate Predictions

Fig. 2: Predictions of the various networks over 300 time steps.

V. Results

Each network was evaluated according to how well it was able to reproduce the literature Lyapunov exponent for it's given attractor. The networks were trained with data pertaining to a Lorenz attractor with $\beta = \mathbf{8/2}, \rho = \mathbf{28}, \sigma = \mathbf{10}$, the literature value of the Lyapunov exponent corresponding to these parameters is $\mathbf{0.9056}$.

References

[1] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey", IEEE Transactions on Neural Networks and Learning Systems **28**, 2222–2232 (2017).

[2] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks", in 2014 IEEE international conference on acoustics, speech and signal processing (ICASSP) (IEEE, 2014), pp. 2164–2168.

[3] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description", in Proceedings of the IEEE conference on computer vision and pattern recognition (2015), pp. 2625–2634.

[4] A. Graves, N. Jaitly, and A.-r. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM", in 2013 IEEE workshop on automatic speech recognition and understanding (IEEE, 2013), pp. 273–278.

[5] Y. Tian and L. Pan, "Predicting short-term traffic flow by long short-term memory recurrent neural network", in 2015 IEEE international conference on smart city/socialcom/sustaincom (smartcity) (IEEE, 2015), pp. 153–158.

[6] E. N. Lorenz, "Deterministic nonperiodic flow", Journal of atmospheric sciences **20**, 130–141 (1963).

[7] J. Pathak, A. Wikner, R. Fussell, S. Chandra, B. R. Hunt, M. Girvan, and E. Ott, "Hybrid forecasting of chaotic processes: using machine learning in conjunction with a knowledge-based model", Chaos: An Interdisciplinary Journal of Nonlinear Science **28**, 041101 (2018).

[8] J. Pathak, Z. Lu, B. R. Hunt, M. Girvan, and E. Ott, "Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data", Chaos: An Interdisciplinary Journal of Nonlinear Science **27**, 121102 (2017).

[9] Z. Lu, B. R. Hunt, and E. Ott, "Attractor reconstruction by machine learning", Chaos: An Interdisciplinary Journal of Nonlinear Science **28**, 061104 (2018).

[10] D. Canaday, A. Pomerance, and D. J. Gauthier, "Model-free control of dynamical systems with deep reservoir computing", arXiv preprint arXiv:2010.02285 (2020).

[11] S. Bompas, B. Georgeot, and D. Guéry-Odelin, "Accuracy of neural networks for the simulation of chaotic dynamics: precision of training data vs precision of the algorithm", Chaos: An Interdisciplinary Journal of Nonlinear Science **30**, 113118 (2020).