Exploration of Long Short-Term Memory Networks for Time Series Forecasting

Eliot Liucci

Department of Mathematical Sciences
Montana State University

Date of completion here as Month Day, Year

A writing project submitted in partial fulfillment
of the requirements for the degree

Master of Science in Statistics

APPROVAL

of a writing project submitted by

Eliot Liucci

This writing project has been read by the writing project advisor and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the Statistics Faculty.

Date	Dr. John Smith Writing Project Advisor
Date	Dr. Katharine Banner Writing Projects Coordinator

Abstract

Long Short-Term Memory (LSTM) networks are a special case of Recurrent Neural Networks (RNN) that allow for past information to make informed predictions. LSTMs are able to do this without the exploding/vanishing gradient problem that can occur with RNNs. The data used were collected within the Everglades National Park as water depth time series. In this paper, a comparison of LSTMs is made to other methods such as ARIMA models and Holt-Winters models.

Contents

1 Introduction

Artificial intelligence, also referred to as statistical learning or machine learning, has become the buzz word of modern computing. While the inner workings of most machine learning techniques are overlooked by many disciplines, the end results speak for themselves. Machine learning techniques allow for adaptable models to be applied to a variety of problems. Traditional statistical methods require a deep understanding of the mathematics behind the algorithm, which can lead to an increase in the time to obtain a result. Machine learning algorithms do not require as deep of an understanding, but the reduced level of understanding required comes at the cost of interpretability. Parameters within models like Least Squares Regression can be interpreted directly, allowing for researchers to obtain a deeper understanding of what the model achieves.

Neural networks, sometimes called multilayer perceptrons, will be the focus of this paper. These particular models involve an input layer, a hidden state, and an output layer. A visualization of a nueral network is shown in Figure 1.

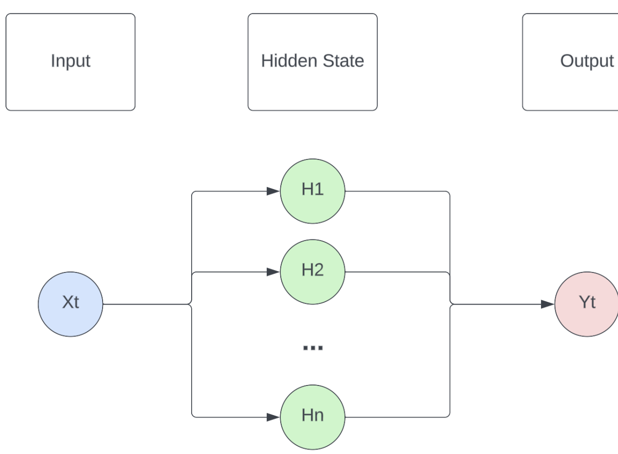

Figure 1: A flowchart depicting a single-layer Neural Network. The input layer (blue) contains a single node, indicating a single value would be taken as an input. The hidden state (green) contains a single layer of n nodes. The output layer (red) contains a single node, indicating a single value would be received as an output. Graphic made with LucidChart.

The nodes within a neural network, specifically within the hidden layer, take an input value from the input layer and apply an activation function, which typically convert any given input to a value within a given range based on the activation function. A visualization is shown for the Linear, Rectified Linear Unit (ReLU), Sigmoid, and Hyperbolic Tangent (Tanh) activation functions in Figure 2.

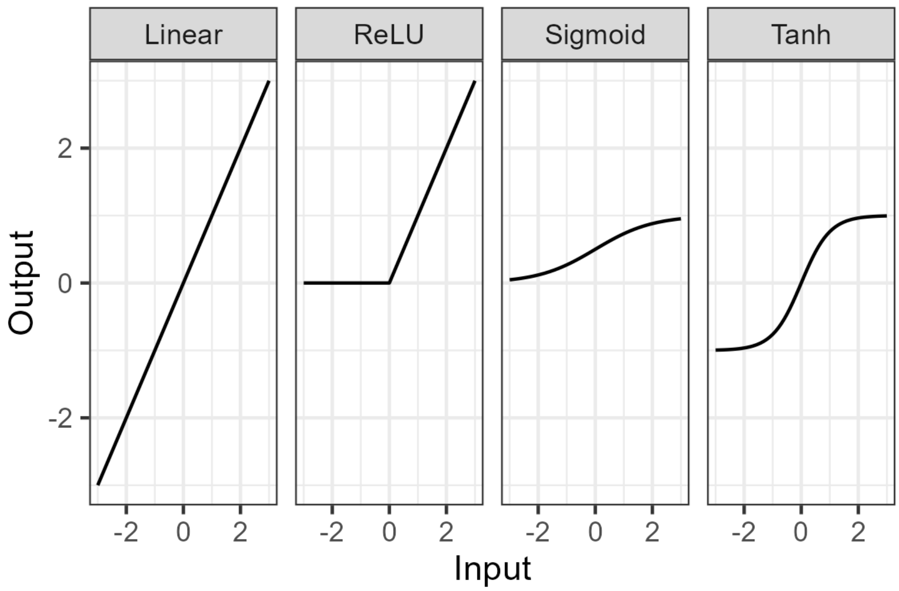

Figure 2: A comparison of common activation functions. The purpose of this visualization is to show how the range of the inputs compares to the range of the outputs for different activations.

In addition to activation functions, each connection between nodes has a weight and bias. For example, the connection between H_1 and X_1 in Figure 1 would take the form:

$$H_1 = \sigma[w_1 * X_1 + b_1]$$

where $\sigma[]$ indicates the activation function being applied, w_1 indicates the weight, and b_1 indicates the bias. The weights and biases in a network are what get tuned in the training process.

Training of neural networks occurs through the process of backpropogation, which is an application of the chain rule in calculus to determine how the output changes based on the change of a single parameter. The prediction of y is compared to the observed value of y in order to approximate a gradient, where each dimension represents a parameter. Then, gradient descent is used to find a minimum error, or loss, in the predictions.

1.1 Recurrent Neural Networks

Recurrent neural networks, or RNNs for short, are essentially multiple neural networks that feed into each other a specified number of times (Rumelhart et al., 1986). Similar to traditional neural networks, these networks have an input layer, a hidden state, and an output layer. As recurrent patterns can be difficult to visualize, these networks are often visualize in an "unrolled" state. This is visualized in Figure 3, where instead of the network looping back into itself 3 times, the 3 different time states are visualized as separate networks.

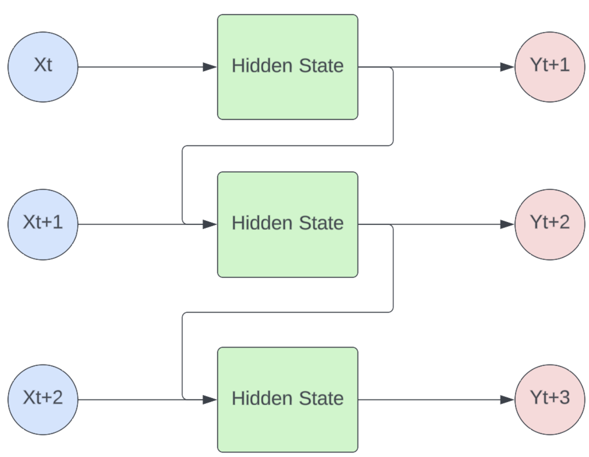

Figure 3: A visualization of an "unrolled" recurrent neural network. This specific example uses 3 time-points of information to get a prediction.

Recurrent neural networks are very useful when working with time series as observations depend on previous observations. A key issue that can arise when using too many time points to predict a single output is an exploding/vanishing gradient, which occurs when a weight associated with time point 1 ends up being multiplied by itself n times to predict n time points ahead. Using Figure 3 as an example network, the prediction for Y_{t+1} would look like:

$$y_1 = \sigma[w_1 \cdot x_1 + b_1] \cdot w_2 + b_2$$

Then, using that prediction in the following time points prediction:

$$y_2 = \sigma[w_1 \cdot x_2 + w_3 \cdot \sigma[w_1 \cdot x_1 + b_1] + b_1] \cdot w_2 + b_2$$

Similarly, for the final prediction:

$$y_3 = \sigma[w_1 \cdot x_3 + w_3 \cdot \sigma[w_1 \cdot x_2 + w_3 \cdot \sigma[w_1 \cdot x_1 + b_1] + b_1] + b_1] \cdot w_2 + b_2$$

In the equations above, w_1 represents the weight applied to the input, w_2 represents the weight applied to the output of the hidden state, and w_3 represents the weight applied to past predictions in the current prediction. Similarly, b_1 represents the bias being added to the input and b_2 represents the bias added to the hidden layer's output. The notation of $\sigma[]$ indicates the activation being applied to whatever is within the brackets. If w_3 is a value above 1, this causes the value of x_1 to have a much larger impact on the final prediction than other inputs. To solve this, "memory" can be added to recurrent neural networks.

1.2 Long Short-Term Memory Networks

Long short-term memory networks are very similar to recurrent neural networks, but they address the issue of gradient approximation by introducing memory cells and a cell state (Hochreiter and Schmidhuber, 1997). A visualization of these networks is shown in Figure 4. The cell state acts as the "long term" memory, making use of all previous information. The only increase that can occur within the cell state is additive, which limits the impact of long-term dependencies that caused issues within RNNs. The memory cells act as the "short term" memory, making use of mostly the current input and previous output.

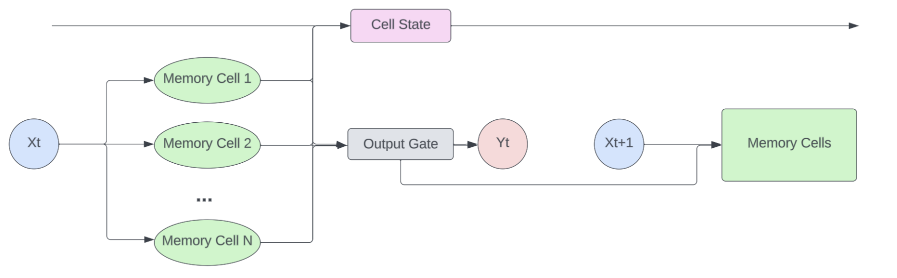

Figure 4: A visualization of long short-term memory networks.

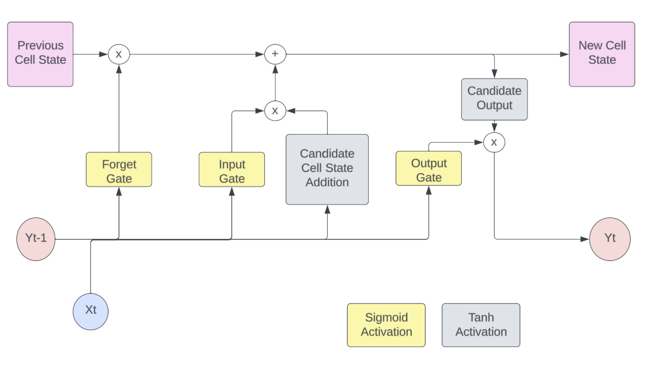

Figure 5: A visualization of the memory cells in LSTMs.

The memory cells are more involved than the typical hidden states of RNNs (see Figure 5). The current input x_t and previous output y_{t-1} are used in more ways than simply getting the next prediction. The "forget gate" determines what percentage of the previous cell state is remembered. By using a sigmoid activation function, the forget gate value will be between 0 and 1. The cell state is then modified based on the candidate cell state addition, which is then multiplied by the input gate value. This gate acts similarly to the forget gate in the sense that it determines what percentage of the candidate cell state addition is actually added to the cell state. Finally, the cell state is fed into a hyperbolic tangent activation and multiplied by the output gate to determine that final output value y_t.

Mathematically, the memory cell can be written in notation used previously (adapted from colah (2015)). The forget gate makes use of a sigmoid activation and has its own corresponding weights and bias, denoted W_f and b_f, respectively.

$$f_t = \sigma[W_f(y_{t-1}, x_t) + b_f]$$

The input gate functions similarly, with the notation of a subscript i.

$$i_t = \sigma[W_i(y_{t-1}, x_t) + b_i]$$

The new candidate values to be added to the cell state use a hyperbolic tangent activation with the subscript c on the weights and bias.

$$\tilde{C}_t = tanh[W_c(y_{t-1}, x_t) + b_c]$$

The updated cell state is the product of the forget gate and the previous cell state plus the new candidate cell state values multiplied by the input gate.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Similar to previously mentioned gates, the output gate uses a sigmoid activation with the subscript o on the weights and bias.

$$o_t = \sigma[W_o(y_{t-1}, x_t) + b_o]$$

The output value y_t is calculated as a proportion (determined by the output gate) of the new cell state.

$$y_t = o_t \cdot tanh[C_t]$$

1.3 EVER Data

In order to train a complex network like LSTMs, a large data set is required. Ideally, the data would have low missingness as missing values can impact training effectiveness. Additionally, the data should not have any seasonal increase over time. Due to these requirements, the training data were selected to be water depth time series collected within the Everglades National park (EVER). There area 52 monitoring stations present, all with various levels of missingness. The chosen station, P33, had less than 1% missingness and had only a minor increasing trend (see Figure 6).

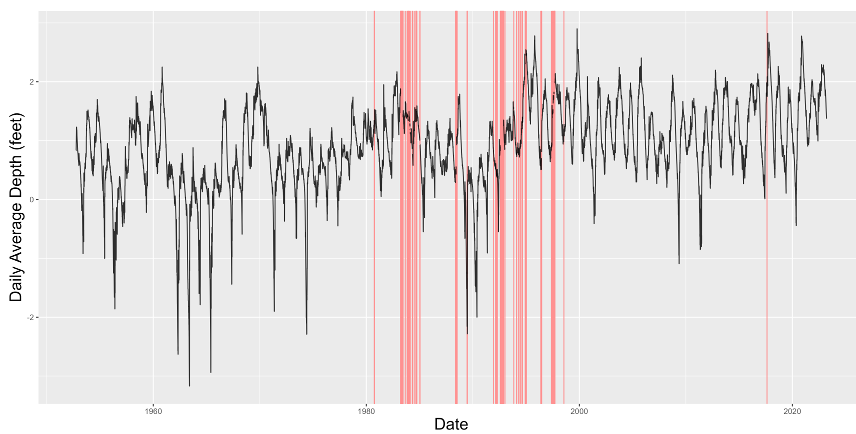

Figure 6: A line plot of the time series used to train the LSTM in this project. Red vertical bars indicate a date with a missing value.

Due to the present although low missingness, an interpolation method had to be chosen to impute those values. Several candidate algorithms were tested, but eventually the decision to use a Kalman filter was made (Kalman, 1960). The line plot in Figure 7 displays the imputed values for the date range of 1960 to 2000.

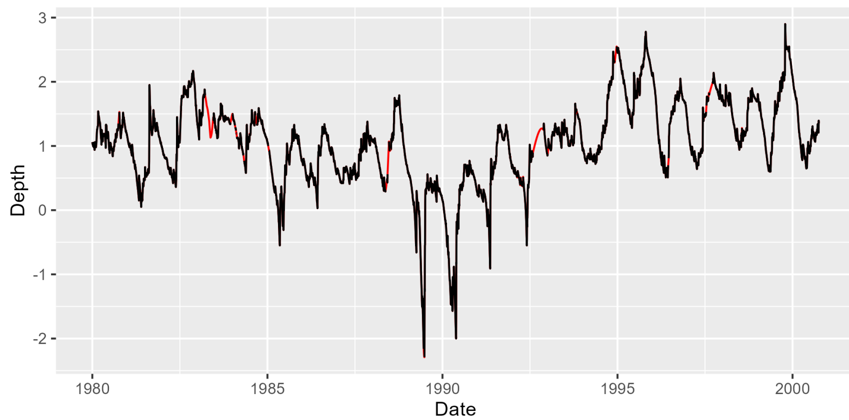

Figure 7: A line plot of the time series used to train the LSTM in this project. Red lines indicate the imputed values using the Kalman filter. Note that this plot focuses on 1960 to 2000, despite imputation being performed across the entire time series.

2 Methods

3 Results

3.1 Reflection

References

Aghabozorgi, S., Shirkhorshidi, A. S., and Wah, T. Y. (2015). Time-series clustering–a decade review. *Information systems*, 53:16–38.

Akusok, A., Björk, K.-M., Miche, Y., and Lendasse, A. (2015). High-performance extreme learning machines: a complete toolbox for big data applications. *IEEE Access*, 3:1011–1025.

colah (2015). Understanding lstm networks.

Everitt, B. and Hothorn, T. (2011). *An introduction to applied multivariate analysis with R*. Springer Science & Business Media.

Hebbar, N. (2021). Time series forecasting with rnn(lstm)— complete python tutorial—.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

James, G., Witten, D., Hastie, T., Tibshirani, R., et al. (2013). *An introduction to statistical learning*, volume 112. Springer.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

Lim, B. and Zohren, S. (2021). Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209.

Maharaj, E. A., D'Urso, P., and Caiado, J. (2019). *Time series clustering and classification*. CRC Press.

Medsker, L. and Jain, L. C. (1999). *Recurrent neural networks: design and applications*. CRC press.

Milligan, G. W. and Cooper, M. C. (1985). An examination of procedures for determining the number of clusters in a data set. *Psychometrika*, 50:159–179.

Prakaisak, I. and Wongchaisuwat, P. (2022). Hydrological time series clustering: A case study of telemetry stations in thailand. *Water*, 14(13):2095.

Rani, S. and Sikka, G. (2012). Recent techniques of clustering of time series data: a survey. *International Journal of Computer Applications*, 52(15).

Ruiz, P. (2019). Ml approached for time seires. *Medium*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71:599–607.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Warren Liao, T. (2005). Clustering of time series data—a survey. *Pattern Recognition*, 38(11):1857–1874.