Modeling COVID-19 Case Counts with LSTM Networks

Brandon Voigt

February 2021

1 Introduction

The goal of this project was to model COVID-19 case counts by using long short-term memory (LSTM) networks, which are a type of recurrent neural network (RNN). The dataset used to train the model included the number of confirmed COVID-19 cases per day for each state in the United States through November 2020. The last 14 days in the dataset were set aside as a test set, in order to evaluate the predictive power of the model over a two-week window.

The first section of this paper reviews the theory behind LSTM networks, beginning with the basics of artificial neural networks, then explaining the concepts of an RNN and how the LSTM architecture is an enhancement of the simple RNN architecture. Next, the dataset used for the project and the chosen model are described in detail. Finally, the results are compared to several other time series prediction methods, including a naive forecast, theta forecast, and ARIMA forecast. The LSTM method performs similarly to the theta method and outperforms the other forecasting methods.

2 Background

An LSTM network is one type of artificial neural network, which is a general class of machine learning models that are loosely inspired by the systems of the human brain. The two basic components of an artificial neural network are nodes that modify an input, and the connections between these nodes [1].

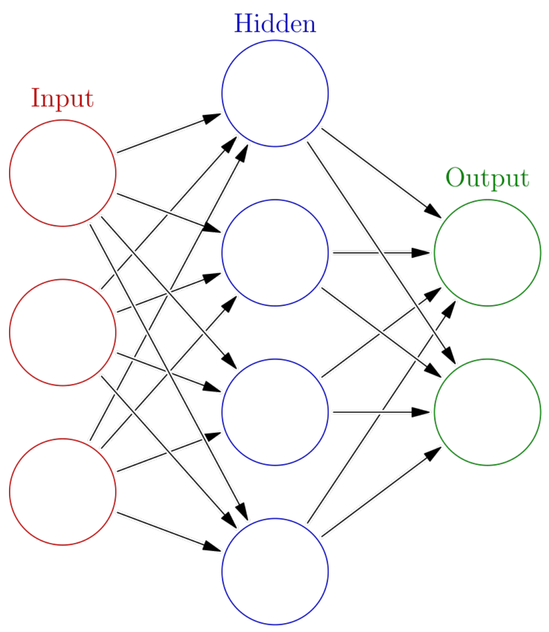

Figure 1: A feedforward artificial neural network with one hidden layer. (Source: en.wikipedia.org)

By analogy, the nodes are like biological neurons that produce a signal, and the connections between nodes are like synapses that pass this signal to other neurons in the network. The architecture of this model can be represented as a directed graph [2], as in Figure 1. Typically, it is organized as a series of layers of nodes, where each layer is only connected to the layers that are adjacent to it. The overall flow of data includes an input layer, one or more “hidden” layers, and a final output layer [3].

Each connection between layers of the network is assigned a weight, such that the linear combination of the outputs from one layer becomes the input to a node in the next layer. The node itself applies a function, which is called an activation function and is usually nonlinear, to this input and gets a new output. This occurs for each node in the layer, and then the process begins again as the outputs from each node are combined and fed to the next layer of the network. The goal of training a neural network, therefore, is to find the weights that produce the most accurate model [4].

Neural networks can be characterized as either feedforward or recurrent. In a feedforward network, each layer is only connected to the following layer,

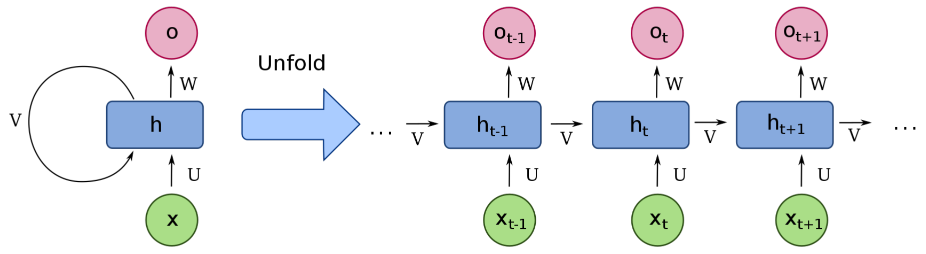

Figure 2: An unfolded recurrent neural network. (Source: en.wikipedia.org)

so the model forms a directed acyclic graph and information flows in one direction from the input layer to the output layer. On the other hand, a recurrent network allows neurons to connect to themselves or to previous layers [5]. One basic RNN structure is called an Elman network, which is also known as a vanilla RNN or simple RNN. This structure includes one recurrent cell, a node that connects to itself. As shown in Figure 2, the system can be "unrolled" such that its structure is similar to that of a feedforward network, and each layer of the unrolled network corresponds to one time step in the recurrent cell [6]. This gives some intuition into why RNNs are well-suited to time series and sequence prediction problems – their structure has an inherent temporal component that aligns with many real-world applications [5].

Following the notation in [6], the hidden state $\boldsymbol{h}[t]$ and output $\boldsymbol{y}[t]$ at timestep t for an Elman RNN can be represented by the following equations:

$$\begin{aligned}\boldsymbol{h}[t] &= f\left(\boldsymbol{W}_i(\boldsymbol{x}[t] + \boldsymbol{b}_i) + \boldsymbol{W}_h(\boldsymbol{h}[t-1] + \boldsymbol{b}_h)\right)\\ \boldsymbol{y}[t] &= g\left(\boldsymbol{W}_o(\boldsymbol{h}[t] + \boldsymbol{b}_o)\right)\end{aligned}$$

where $f(\cdot)$ is the activation function of the hidden node; $g(\cdot)$ is the function that transforms the output of the hidden layer to the final output; $\boldsymbol{W}_i, \boldsymbol{W}_h$, and $\boldsymbol{W}_o$ are weight matrices; $\boldsymbol{x}[t]$ is the input $\boldsymbol{x}$ at timestep t; and $\boldsymbol{b}_i, \boldsymbol{b}_h$, and $\boldsymbol{b}_o$ are bias vectors. In short, the hidden state at a given timestep is a function of both the input at that timestep and itself at the previous timestep, while the output is a function of the hidden state at that timestep.

The most common algorithm for training an RNN is called backpropagation through time [3] [7]. At a high level, this algorithm optimizes the weights of the network by repeatedly applying the chain rule. The process relies on the idea of unrolling an RNN, since an unrolled RNN is equivalent to a feedforward network when a finite number of timesteps are considered. A feedforward network forms a directed acyclic graph, which makes it possible to update the weights recursively based on the error of the network, moving backwards in time [7].

However, applications of the simple RNN frequently encounter an issue called the "vanishing error" or "vanishing gradient" problem. The backpropagated error either grows or shrinks with each timestep, so when the weights are updated over many timesteps, they either oscillate back and forth or change extremely slowly. In practice, this means that a simple RNN typically cannot bridge more than 10 timesteps [3]. A proof that this issue will occur regardless of the large-signal stability of the network can be found in [7].

The LSTM architecture was first proposed in 1997 by Hochreiter and Schmidhuber [8] as a way to counteract the vanishing gradient issue. It enforces a constant error flow by using a system of gates within the recurrent cell, including a "forget gate," "update gate," and "output gate." The equations for an LSTM cell can be written as:

$$\begin{aligned}
\text{forget gate}:\ \sigma_f[t] &= \sigma(\boldsymbol{W}_f\boldsymbol{x}[t] + \boldsymbol{R}_f\boldsymbol{y}[t-1] + \boldsymbol{b}_f)\\
\text{candidate state}:\ \tilde{\boldsymbol{h}}[t] &= g_1(\boldsymbol{W}_h\boldsymbol{x}[t] + \boldsymbol{R}_h\boldsymbol{y}[t-1] + \boldsymbol{b}_h)\\
\text{update gate}:\ \sigma_u[t] &= \sigma(\boldsymbol{W}_u\boldsymbol{x}[t] + \boldsymbol{R}_u\boldsymbol{y}[t-1] + \boldsymbol{b}_u)\\
\text{cell state}:\ \boldsymbol{h}[t] &= \sigma_u[t] \odot \tilde{\boldsymbol{h}}[t] + \sigma_f[t] \odot \boldsymbol{h}[t-1]\\
\text{output gate}:\ \sigma_o[t] &= \sigma(\boldsymbol{W}_o\boldsymbol{x}[t] + \boldsymbol{R}_o\boldsymbol{y}[t-1] + \boldsymbol{b}_o)\\
\text{output}:\ \boldsymbol{y}[t] &= \sigma_o[t] \odot g_2(\boldsymbol{h}[t])
\end{aligned}$$

again following the notation in [6], where $\boldsymbol{x}[t]$ is the input vector at time t; $\boldsymbol{W}_f, \boldsymbol{W}_h, \boldsymbol{W}_u$, and $\boldsymbol{W}_o$ are weight matrices applied to the input; $\boldsymbol{R}_f, \boldsymbol{R}_h, \boldsymbol{R}_u$, and $\boldsymbol{R}_o$ are weight matrices applied to the previous output; $\boldsymbol{b}_f, \boldsymbol{b}_h, \boldsymbol{b}_u$, and $\boldsymbol{b}_o$ are bias vectors; $\sigma(\cdot)$ is the sigmoid function; $g_1(\cdot)$ and $g_2(\cdot)$ are nonlinear activation functions, typically the hyperbolic tangent; and $\odot$ denotes the Hadamard (elementwise) product. See Figure 3 for a diagram of the overall architecture. This system of gates controls the flow of information in and out of the node, and scales the error appropriately. The end result is that an LSTM network can bridge much larger time lags, even more than 1000 timesteps [8].

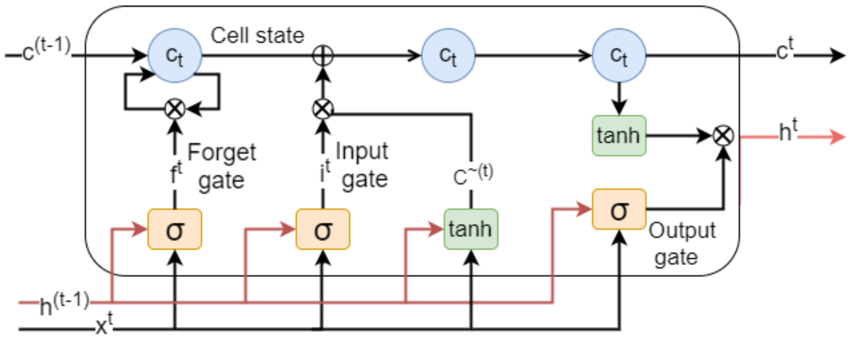

Figure 3: An LSTM cell with forget gate, input gate, and output gate. (Source: researchgate.net)

The LSTM network can be extended further by using multiple LSTM cells, which is known as a deep LSTM or stacked LSTM, as shown in Figure. Deep feedforward networks, which use an architecture with many hidden layers, have been highly successful in fields such as computer vision and machine translation [9], and many of the same concepts can be applied to RNNs. For example, the stacked LSTM structure was shown to achieve state-of-the-art results on TIMIT, a speech recognition dataset [10]. It has also been applied to biological time series [11] [12], where it again achieved better empirical results than other methods. For this project, a stacked LSTM architecture was chosen, which will be described in more detail in the next section.

3 Project Details

The dataset used for this project was collected by BroadStreet Health as part of the COVID-19 Data Project [13]. In its original format, the dataset included the cumulative number of confirmed COVID-19 cases per day for each county in the United States. The first step in the project was to aggregate these county-level totals to the state level, and to difference the dataset to get the number of new cases per day rather than the cumulative total. The most recent 14 days of data, comprising the last two weeks of November 2020, were set aside as a test set. The overall goal, then, was to build a model that would accurately predict the number of new cases in each state over a 14-day period.

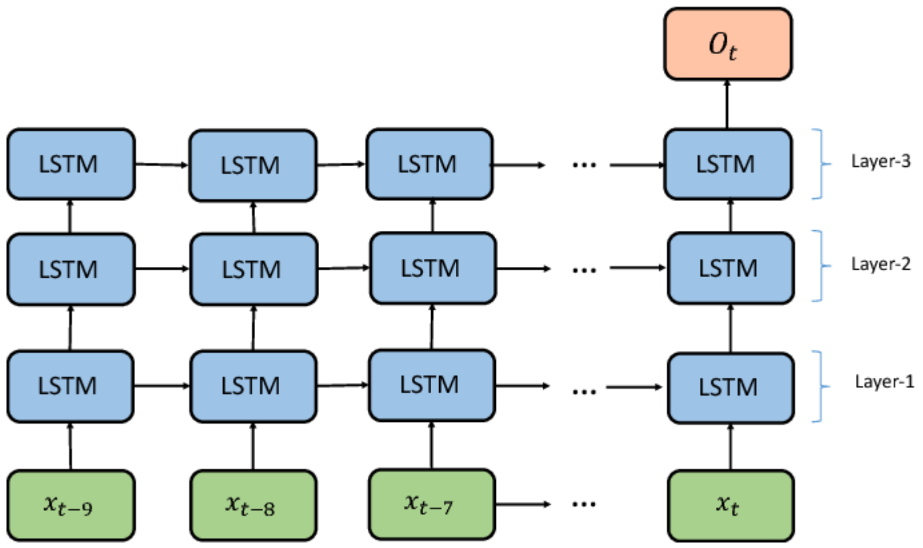

Figure 4: A stacked LSTM network. (Source: semanticscholar.org)

The training dataset included 299 days' worth of data for 52 states and territories (including the District of Columbia and Puerto Rico). This data was transformed into sequences of 180 days to be used as input to the LSTM network, with the number of new cases in the next day as the output. In other words, the model was trained to predict the number of new cases based on the previous 180 days. The 14-day predictions used to evaluate the model were generated recursively, by predicting one day at a time and then incorporating this prediction into the input for the following day.

The chosen architecture was a stacked LSTM network. In comparison to Figure 4, which has 3 layers of LSTM cells, this network included 2 layers of LSTM cells, followed by a fully connected layer. An important consideration when building the model was to prevent overfitting. The number of features in the hidden state $\boldsymbol{h}$ was chosen to be 5, which resulted in better predictions than a larger number of features. Another method that was used to prevent overfitting was "dropout," in which a random subset of nodes and their connections is dropped from the model during training [14]. In this case, a dropout rate of 0.25 was used in between the two LSTM layers. Although other, more complex models were considered, most of them seemed to overfit and did not achieve better prediction accuracy.

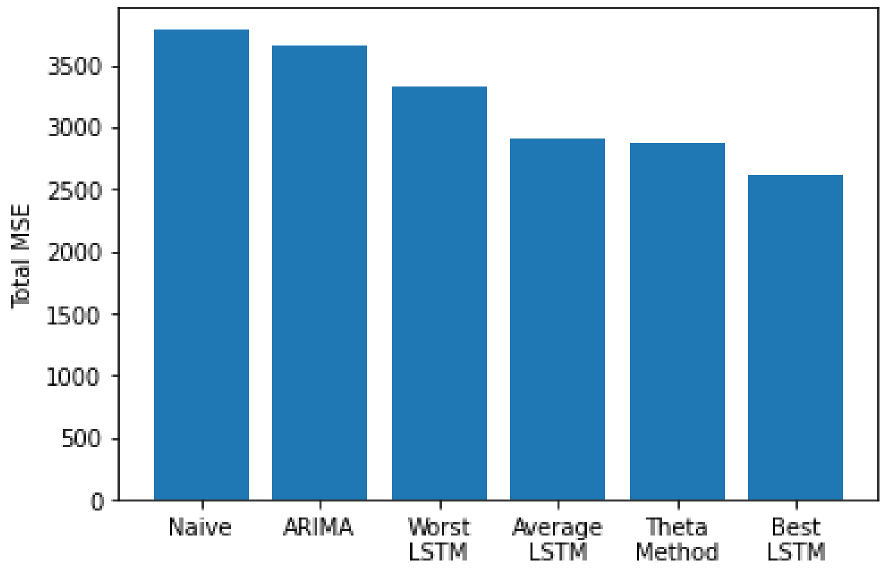

Figure 5: Comparison of the total mean-squared error on the test set for different forecasting methods.

4 Results

To evaluate the accuracy of the stacked LSTM model, we can compare its prediction accuracy to that of other forecasting methods. Three other methods were chosen: a naive forecaster that simply predicted the value of the most recent observation, an autoregressive integrated moving average (ARIMA) forecaster, and a theta forecaster. The results are shown in Figure 5. Since the process of training an LSTM or other neural network introduces randomness, the LSTM network was trained 10 times in a row; its best, worst, and average performance are shown. Although the best LSTM outperformed the theta forecaster, its average performance was slightly worse. On the other hand, all 10 trials of the LSTM network achieved better predictions than the naive and ARIMA forecasters.

Although the LSTM model achieved reasonably accurate results, we can see that it did not outperform the simpler theta method for forecasting. This model might benefit from more training data. It is also possible that a different architecture incorporating an LSTM cell could result in better predictions. One limitation of the LSTM approach, and the other methods used for comparison, is that they are not domain-specific. A model that is specifically designed for epidemiology might work better in this case. Nevertheless, despite the limitations of this approach, it shows that LSTM networks are a viable model for time series prediction. With more fine-tuning, it may be possible to achieve state-of-the-art results with an LSTM-based architecture.

References

[1] Grossi, Enzo, and Massimo Buscema. "Introduction to Artificial Neural Networks." *European Journal of Gastroenterology & Hepatology* 19, No. 12 (2007): 1046–54. https://doi.org/10.1097/meg.0b013e3282f198a0

[2] Dongare, A.D., R.R. Kharde, and Amit D. Kachare. "Introduction to Artificial Neural Network." *International Journal of Engineering and Innovative Technology* Volume 2, Issue 1 (2012).

[3] Staudemeyer, Ralf C., and Eric Rothstein Morris. "Understanding LSTM: a tutorial into Long Short-Term Memory Recurrent Neural Networks" (2019). https://arxiv.org/abs/1909.09586

[4] Sordo, Margarita. "Introduction to Neural Networks in Healthcare" (2002). https://www.researchgate.net/publication/228820949

[5] Hewamalage, Hansika, Christoph Bergmeir, and Kasun Bandara. "Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions." *International Journal of Forecasting* 37, No. 1 (2021): 388–427. https://doi.org/10.1016/j.ijforecast.2020.06.008

[6] Bianchi, Filippo Maria, Enrico Maiorino, Michael C. Kampffmeyer, Antonello Rizzi, and Robert Jenssen. *Recurrent Neural Networks for Short-Term Load Forecasting An Overview and Comparative Analysis.* Cham: Springer International Publishing (2017).

[7] Sherstinsky, Alex. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." *Physica D: Nonlinear Phenomena* 404 (2020). https://doi.org/10.1016/j.physd.2019.132306

[8] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation* 9, No. 8 (1997): 1735–80. https://doi.org/10.1162/neco.1997.9.8.1735

[9] Wang, Haohan, and Bhiksha Raj. “On the Origin of Deep Learning” (2017). https://arxiv.org/abs/1702.07800

[10] Graves, Alex, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech Recognition with Deep Recurrent Neural Networks” (2013). https://arxiv.org/abs/1303.5778

[11] Malhotra, Pankaj, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. “Long Short Term Memory Networks for Anomaly Detection in Time Series” (2015). https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf

[12] Prasad, Sharat C., and Piyush Prasad. “Deep Recurrent Neural Networks for Time Series Prediction” (2014). https://arxiv.org/abs/1407.5949

[13] “COVID-19 Data Project.” BroadStreet Health (2021). http://covid19dataproject.org/data

[14] Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” *Journal of Machine Learning Research* 15 (2014) 1929-1958. https://jmlr.org/papers/v15/srivastava14a.html