Long Short-Term Memory Model for Modeling COVID-19 Cases in Tokyo, Japan

Zen Tamura

1. Model Overview

First introduced by Hochreiter & Schmidhuber (1997) [1], and further modified by Gers et al. (2000) [2], Long Short-Term Memory networks (LSTMs) are a type of recurrent neural network (RNN) that addresses the vanishing gradient problem prevalent in RNNs. Here, we used a simple LSTM network with a single hidden layer, an input layer, and an output layer. The model was trained to estimate the number of new COVID-19 cases for the following day, given some number of past cases as input.

2. Data Sources and Preparation

We used the number of daily confirmed COVID-19 cases in Tokyo, Japan from January 24, 2020 to December 8, 2020. The data is publicly available via the Tokyo Metropolitan government's website. [3]

The data was first split into a training and validation set. Confirmed cases from January 24, 2020 to October 31, 2020 was used for the training set. Data from November 1, 2020 to December 8, 2020 was used for the validation set. Before training, the data was scaled to $[-1, 1]$ using normalization. Formally, given an input sequence $\mathbf{x}_t$ for time t, each entry $x^{(i)} \in \mathbf{x}_t$ was scaled according to

$$x^{(i)}_{\text{scaled}} = \frac{x^{(i)} - \min(\mathbf{x}_t)}{\max(\mathbf{x}_t) - \min(\mathbf{x}_t)} \in [-1, 1],\ \forall i. \tag{1}$$

3. Forward Propagation Steps

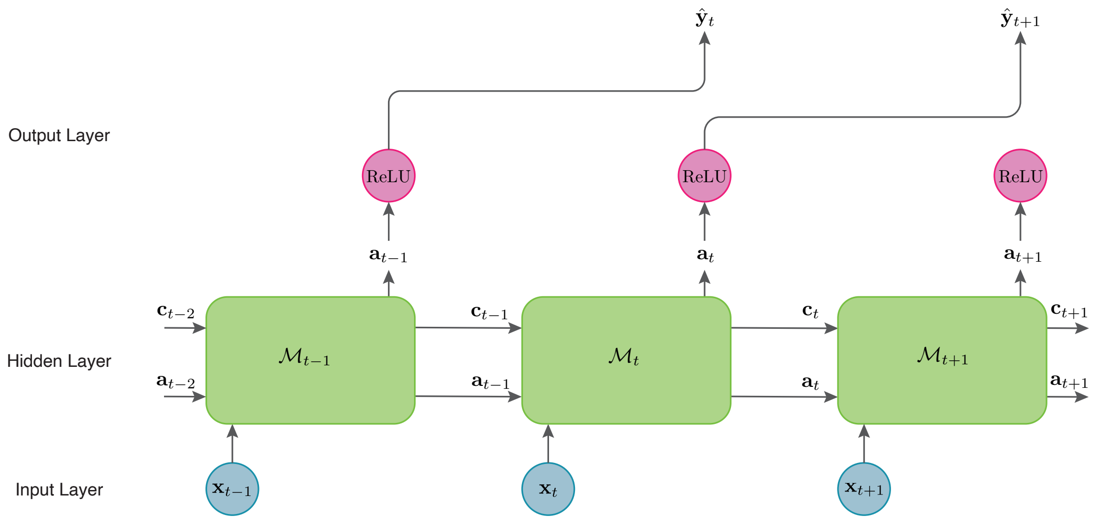

Figure 1: LSTM architecture diagram.

The entire LSTM model architecture is graphically represented in Figure 1. It comprises of an input layer, one hidden layer with LSTM cells connected sequentially, and an output layer. A single LSTM cell at time t, $\mathcal{M}_t$, is a mathematical function that takes three inputs and returns two outputs:

$$(\mathbf{a}_t, \mathbf{c}_t) = \mathcal{M}_t(\mathbf{a}_{t-1}, \mathbf{c}_{t-1}, \mathbf{x}_t) \tag{2}$$

where $\mathbf{a}_t, \mathbf{a}_{t-1} \in \mathbb{R}^{n_a}$ are the hidden states, $\mathbf{c}_t, \mathbf{c}_{t-1} \in \mathbb{R}^{n_a}$ are the cell states, and $\mathbf{x}_t \in \mathbb{R}^{n_x}$ is the inputs at time t. The inputs $\mathbf{x}_t$ were created by sliding a moving window of length n_x across the entire dataset. In other words, the number of confirmed cases from time $t - n_x$ to $t - 1$ were used for inputs at time t. This also gives us $T - n_x$ cells and inputs, when T is the total number of data points. We tuned the parameter n_a, n_x using random search [4], which we discuss in Section 5.

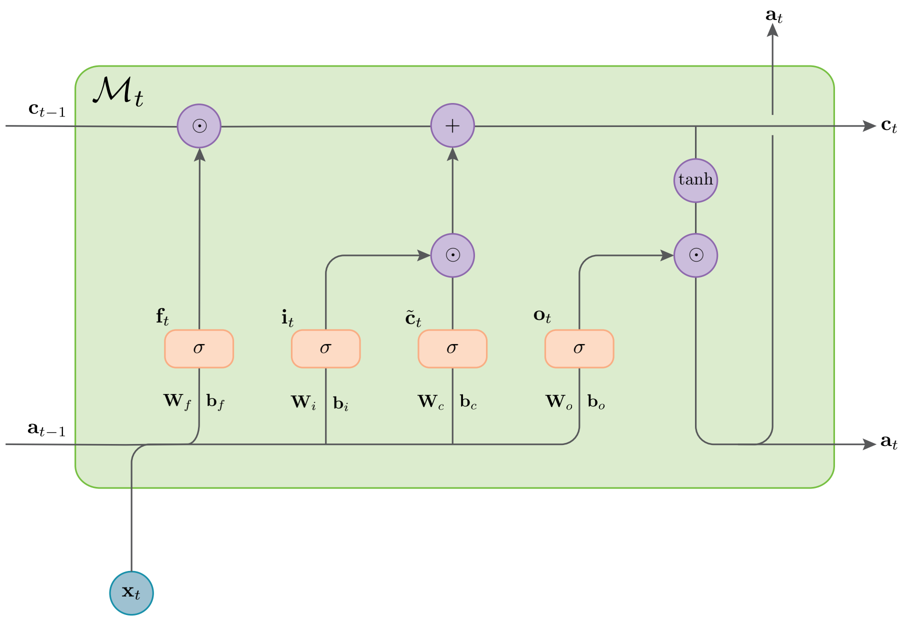

Figure 2: LSTM cell diagram.

Inside the cell, the inputs are passed through four functions (gates): the forget gate $\mathbf{f}_t$, the input gate $\mathbf{i}_t$, the output gate $\mathbf{o}_t$, and the cell update $\tilde{\mathbf{c}}_t$. (Figure 2) For convenience, we denote $\mathbf{z}_t = \begin{bmatrix} \mathbf{a}_{t-1} \\ \mathbf{x}_t \end{bmatrix} \in \mathbb{R}^{(n_a+n_x)}$. Each gate is represented as:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{z}_t + \mathbf{b}_f) = \sigma(\mathbf{l}_f) \in \mathbb{R}^{n_a}, \tag{3}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{z}_t + \mathbf{b}_i) = \sigma(\mathbf{l}_i) \in \mathbb{R}^{n_a}, \tag{4}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{z}_t + \mathbf{b}_o) = \sigma(\mathbf{l}_o) \in \mathbb{R}^{n_a}, \tag{5}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c \mathbf{z}_t + \mathbf{b}_c) = \tanh(\mathbf{l}_c) \in \mathbb{R}^{n_a}. \tag{6}$$

Here, $\mathbf{W}_f, \mathbf{W}_i, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{n_a \times (n_a+n_x)}$ are the weight matrices and $\mathbf{b}_f, \mathbf{b}_i, \mathbf{b}_o, \mathbf{b}_c \in \mathbb{R}^{n_a}$ are the biases. The weights and biases are shared across all time steps and are learned through backpropagation. $\sigma(\cdot)$ is the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$. Based on these functions, the cell state at time t is updated according to

$$\mathbf{c}_t = \mathbf{i}_t \odot \tilde{\mathbf{c}}_t + \mathbf{f}_t \odot \mathbf{c}_{t-1} \in \mathbb{R}^{n_a}. \tag{7}$$

$\odot$ is the element-wise product, or the Hadamard product of matrices. The forget gate $\mathbf{f}_t$ controls how much information stored in the previous cell state to forget, and the input gate $\mathbf{i}_t$ dictates how much of the input gained through the candidate value $\tilde{\mathbf{c}}_t$ to incorporate in the new cell state.

The hidden state $\mathbf{a}_t$ is also updated according to

$$\mathbf{a}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \in \mathbb{R}^{n_a}. \tag{8}$$

The output gate $\mathbf{o}_t$ decides how much information should come into the next hidden state from the cell state.

The hidden state is then passed onto the output layer, where the predictions for time $t + 1$ are computed as

$$\hat{\mathbf{y}}_{t+1} = \text{ReLU}(\mathbf{W}_y \mathbf{a}_t + \mathbf{b}_y) = \text{ReLU}(\mathbf{v}_t) \in \mathbb{R}^{n_y} \tag{9}$$

where $\text{ReLU}(z) = \max(0, z)$ and $\mathbf{W}_y \in \mathbb{R}^{n_y \times n_a}$, $\mathbf{b}_y \in \mathbb{R}^{n_y}$. Our LSTM therefore requires a total of $n_a(4n_a + 4n_x + n_y)$ weights and $4n_a + n_y$ biases. For our model, n_y was fixed to 1.

4. Backpropagation Steps

To learn the optimal parameters $\mathbf{W}^*_\star, \mathbf{b}^*_\star$ for $\star = \{f, i, o, c, y\}$ such that

$$\mathbf{W}^*_\star, \mathbf{b}^*_\star = \arg\min_{\mathbf{W}_\star, \mathbf{b}_\star} J \tag{10}$$

we used Adam [5] as our optimization algorithm with a learning rate of $\alpha = 0.001$, and hyperparameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-8}$. The model was trained for at least 200 iterations of Adam, but we stopped training when the difference in loss from the previous iteration was smaller than 7×10^{-9}. Weights $\mathbf{W}_\star$ were initialized to small random values, where $\mathbf{W}_\star \sim \mathcal{N}(0, 1) \times 0.01$. Biases $\mathbf{b}_\star$ were initialized to zero. While training, gradients were clipped to $[-1, 1]$ to prevent exploding gradients.

Adam, an optimization algorithm first proposed by Kingma & Ba (2015) [5], is described in full below. Note $\star = \{f, i, o, c, y\}$.

Algorithm 1 Adam

```
Require: $\alpha$ : learning rate
Require: $\beta_1, \beta_2 \in [0, 1)$
Require: $\varepsilon$: float
1: $V_{dW} = 0,\ V_{db} = 0,\ S_{dW} = 0,\ S_{db} = 0$
2: while not converged do
3:     $t \leftarrow t + 1$
4:     $V_{dW} = \beta_1 V_{dW} + (1 - \beta_1)\frac{\partial J}{\partial \mathbf{W}_\star},\ V_{db} = \beta_1 V_{db} + (1 - \beta_1)\frac{\partial J}{\partial \mathbf{b}_\star}$
5:     $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2)\frac{\partial J}{\partial \mathbf{W}_\star} \odot \frac{\partial J}{\partial \mathbf{W}_\star},\ S_{db} = \beta_2 S_{db} + (1 - \beta_2)\frac{\partial J}{\partial \mathbf{b}_\star} \odot \frac{\partial J}{\partial \mathbf{b}_\star}$
6:     $\widehat{V_{dW}} = \frac{V_{dW}}{(1 - \beta_1^t)},\ \widehat{V_{db}} = \frac{V_{db}}{(1 - \beta_1^t)}$
7:     $\widehat{S_{dW}} = \frac{S_{dW}}{(1 - \beta_2^t)},\ \widehat{S_{db}} = \frac{S_{db}}{(1 - \beta_2^t)}$
8:     $\mathbf{W}_\star \leftarrow \mathbf{W}_\star - \alpha\frac{\widehat{V_{dW}}}{\sqrt{\widehat{S_{dW}} + \varepsilon}},\ \mathbf{b}_\star \leftarrow \mathbf{b}_\star - \alpha\frac{\widehat{V_{db}}}{\sqrt{\widehat{S_{db}} + \varepsilon}}$
9: end while
```

In the following paragraphs, we detail the backpropagation steps used to calculate the gradients $\frac{\partial J}{\partial \mathbf{W}_\star}, \frac{\partial J}{\partial \mathbf{b}_\star}$ used in Adam. First, we calculate

$$\frac{\partial J}{\partial \mathbf{v}_t} = \frac{1}{2m}(\hat{y}_i - y_i)(\mathbf{1}\{\mathbf{v}_t \geq 0\} - y_i)$$

where $\mathbf{1}\{\cdot\}$ denotes the indicator function: $\mathbf{1}\{\text{True}\} = 1$, $\mathbf{1}\{\text{False}\} = 0$. Using this and the chain rule of calculus, we obtain the gradients for each of the weights and biases.

(1) Output layer

$$\frac{\partial J}{\partial \mathbf{W}_y} = \frac{\partial J}{\partial \mathbf{v}_t}\frac{\partial \mathbf{v}_t}{\partial \mathbf{W}_y} = \frac{\partial J}{\partial \mathbf{v}_t}\mathbf{a}_t^T$$

$$\frac{\partial J}{\partial \mathbf{b}_y} = \frac{\partial J}{\partial \mathbf{v}_t}\frac{\partial \mathbf{v}_t}{\partial \mathbf{b}_y} = \frac{\partial J}{\partial \mathbf{v}_t}$$

(2) Hidden state

$$\frac{\partial J}{\partial \mathbf{a}_t} = \frac{\partial J}{\partial \mathbf{v}_t}\frac{\partial \mathbf{v}_t}{\partial \mathbf{a}_t} = \mathbf{W}_y^T \frac{\partial J}{\partial \mathbf{v}_t}$$

(3) Output gate

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{o}_t} &= \frac{\partial J}{\partial \mathbf{a}_t}\frac{\partial \mathbf{a}_t}{\partial \mathbf{o}_t} = \frac{\partial J}{\partial \mathbf{a}_t} \odot \tanh(\mathbf{c}_t) \\
\frac{\partial J}{\partial \mathbf{l}_o} &= \frac{\partial J}{\partial \mathbf{o}_t}\frac{\partial \mathbf{o}_t}{\partial \mathbf{l}_o} = \frac{\partial J}{\partial \mathbf{o}_t} \odot \mathbf{l}_o \odot (1 - \mathbf{l}_o) \\
\frac{\partial J}{\partial \mathbf{W}_o} &= \frac{\partial J}{\partial \mathbf{l}_o}\frac{\partial \mathbf{l}_o}{\partial \mathbf{W}_o} = \frac{\partial J}{\partial \mathbf{l}_o}\mathbf{z}_t^T \\
\frac{\partial J}{\partial \mathbf{b}_o} &= \frac{\partial J}{\partial \mathbf{l}_o}\frac{\partial \mathbf{l}_o}{\partial \mathbf{b}_o} = \frac{\partial J}{\partial \mathbf{l}_o}
\end{aligned}$$

(4) Cell state

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{c}_t} &= \frac{\partial J}{\partial \mathbf{a}_t}\frac{\partial \mathbf{a}_t}{\partial \mathbf{c}_t} = \frac{\partial J}{\partial \mathbf{a}_t} \odot \mathbf{o}_t \odot (1 - \tanh^2(\mathbf{c}_t)) \\
\frac{\partial J}{\partial \tilde{\mathbf{c}_t}} &= \frac{\partial J}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \tilde{\mathbf{c}_t}} = \frac{\partial J}{\partial \mathbf{c}_t} \odot \mathbf{i}_t \\
\frac{\partial J}{\partial \mathbf{l}_c} &= \frac{\partial J}{\partial \tilde{\mathbf{c}_t}}\frac{\partial \tilde{\mathbf{c}_t}}{\partial \mathbf{l}_c} = \frac{\partial J}{\partial \tilde{\mathbf{c}_t}} \odot (1 - \tanh^2(\mathbf{l}_c)) = \frac{\partial J}{\partial \tilde{\mathbf{c}_t}} \odot (1 - \tilde{\mathbf{c}_t}^2) \\
\frac{\partial J}{\partial \mathbf{W}_c} &= \frac{\partial J}{\partial \mathbf{l}_c}\frac{\partial \mathbf{l}_c}{\partial \mathbf{W}_c} = \frac{\partial J}{\partial \mathbf{l}_c} z_t^T \\
\frac{\partial J}{\partial \mathbf{b}_c} &= \frac{\partial J}{\partial \mathbf{l}_c}\frac{\partial \mathbf{l}_c}{\partial \mathbf{b}_c} = \frac{\partial J}{\partial \mathbf{l}_c}
\end{aligned}$$

(5) Input gate

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{i}_t} &= \frac{\partial J}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{i}_t} = \frac{\partial J}{\partial \mathbf{c}_t} \odot \tilde{\mathbf{c}}_t \\
\frac{\partial J}{\partial \mathbf{l}_i} &= \frac{\partial J}{\partial \mathbf{i}_t}\frac{\partial \mathbf{i}_t}{\partial \mathbf{l}_i} = \frac{\partial J}{\partial \mathbf{i}_t} \odot \mathbf{l}_i \odot (1 - \mathbf{l}_i) \\
\frac{\partial J}{\partial \mathbf{W}_i} &= \frac{\partial J}{\partial \mathbf{l}_i}\frac{\partial \mathbf{l}_i}{\partial \mathbf{W}_i} = \frac{\partial J}{\partial \mathbf{l}_i} z_t^T \\
\frac{\partial J}{\partial \mathbf{b}_i} &= \frac{\partial J}{\partial \mathbf{l}_i}\frac{\partial \mathbf{l}_i}{\partial \mathbf{b}_i} = \frac{\partial J}{\partial \mathbf{l}_i}
\end{aligned}$$

(6) Forget gate

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{f}_t} &= \frac{\partial J}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{f}_t} = \frac{\partial J}{\partial \mathbf{c}_t} \odot \mathbf{c}_{t-1} \\
\frac{\partial J}{\partial \mathbf{l}_f} &= \frac{\partial J}{\partial \mathbf{f}_t}\frac{\partial \mathbf{f}_t}{\partial \mathbf{l}_f} = \frac{\partial J}{\partial \mathbf{f}_t} \odot \mathbf{l}_f \odot (1 - \mathbf{l}_f) \\
\frac{\partial J}{\partial \mathbf{W}_f} &= \frac{\partial J}{\partial \mathbf{l}_f}\frac{\partial \mathbf{l}_f}{\partial \mathbf{W}_f} = \frac{\partial J}{\partial \mathbf{l}_f} z_t^T \\
\frac{\partial J}{\partial \mathbf{b}_f} &= \frac{\partial J}{\partial \mathbf{l}_f}\frac{\partial \mathbf{l}_f}{\partial \mathbf{b}_f} = \frac{\partial J}{\partial \mathbf{l}_f}
\end{aligned}$$

5. Hyperparameter Search

We used random search [4] to tune hyperparameters n_a and n_x. n_a and n_x were sampled from discrete uniform distributions $n_a \sim U(16, 128)$, $n_x \sim U(5, 25)$, respectively. We ran a total of 1,000 random search trials to choose the set of optimal parameters (n_a^*, n_x^*) that minimized the validation set error:

$$n_a^*, n_x^* = \arg\min_{n_a, n_x} J_{\text{validation}}. \tag{11}$$

where $J_{\text{validation}}$ is the mean squared error for the validation set. Formally, given the daily reported cases $\mathbf{y} = [y^{(1)}, \ldots, y^{(m)}] \in \mathbb{R}^m$ and the model's estimations for daily cases $\hat{\mathbf{y}} = [\hat{y}^{(1)}, \ldots, y^{(m)}] \in \mathbb{R}^m$, the mean squared error is given as

$$J_{\text{validation}} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2.$$

6. Implementation Details

The LSTM model was implemented with Python 3.7.6 and Numpy 1.19.1. We did not use deep learning libraries such as Tensorflow or Keras for the LSTM implementation. We trained the LSTM and ran the random trials on one NVIDIA GeForce RTX 2080 Ti GPU.

References

[1] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Computation* **9** 8, pp.1735–1780 (1997). https://doi.org/10.1162/neco.1997.9.8.1735

[2] Gers, F.A., Schmidhuber, J.A. & Cummins, F.A. Learning to forget: Continual prediction with LSTM. *Neural Computation* **12** 10 (2000). https://doi.org/10.1162/089976600300015015

[3] Tokyo Metropolitan Government (2020). COVID-19–The Information Website. https://stopcovid19.metro.tokyo.lg.jp/en

[4] Bergstra, J. & Bengio, Y. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* **13**, pp. 281-305 (2012).

[5] Kingma, D.P. & Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations* (eds Bengio, Y. & Lecun, Y.) (2020). https://arxiv.org/abs/1412.6980