Given a sequence of stock prices for three days, we want to use an LSTM to predict the stock price for the next day. Our input vector x will have three elements (x_1, x_2, x_3) representing the stock prices for these three days.

The input vector, x_t:

$$x = \begin{bmatrix} 100 \\ 105 \\ 110 \end{bmatrix}$$

In the case of our stock program, this vector would contain the 60 previous closing days, although to simplify the process, we will use 3 here

Assume we have a single LSTM layer with a hidden state size of 2. We will use made-up values for the weights and biases for illustration purposes.

$$W_f = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, b_f = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}$$

$$W_i = \begin{bmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{bmatrix}, b_i = \begin{bmatrix} 1.1 \\ 1.2 \end{bmatrix}$$

$$W_o = \begin{bmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \end{bmatrix}, b_o = \begin{bmatrix} 1.7 \\ 1.8 \end{bmatrix}$$

$$W_C = \begin{bmatrix} 1.9 & 2.0 \\ 2.1 & 2.2 \end{bmatrix}, b_C = \begin{bmatrix} 2.3 \\ 2.4 \end{bmatrix}$$

The LSTM equations are given by:

$$\begin{aligned} f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ \tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\ h_t &= o_t \odot \tanh(C_t) \end{aligned}$$

1. f_t: Forget gate - This controls how much information from the previous cell state (C_{t-1}) should be kept or forgotten. It is an element-wise multiplication, and its values are between 0 and 1, which determines how much of each component in the cell state should be retained.

2. i_t: Input gate - This controls how much of the new candidate cell state ($\tilde{C}_t$) should be added to the updated cell state (C_t). Like the forget gate, its values are between 0 and 1, which allows the LSTM to control the degree of contribution from the candidate cell state.

3. o_t: Output gate - This controls the proportion of information from the updated cell state (C_t) that should be passed on to the hidden state (h_t). The output gate values are also between 0 and 1, allowing the LSTM to modulate the information that is passed to the next layer or the next time step.

4. $\tilde{C}t$: Candidate cell state - This represents the new information that the LSTM cell can store in its cell state. It is a combination of the input vector (x_t) and the previous hidden state ($ht - 1$). The candidate cell state values can range between -1 and 1 due to the tanh activation function.

5. C_{t-1}: Previous cell state - This stores the information learned in the previous time step. The LSTM cell uses the forget gate (f_t) to decide how much of this information to keep or discard.

6. C_t: Updated cell state - This is the final cell state after combining the information from the previous cell state (C_{t-1}) and the candidate cell state ($\tilde{C}_t$). The input gate (i_t) controls how much of the candidate cell state contributes to the updated cell state.

7. h_{t-1}: Previous hidden state - This is the output from the previous time step that is fed back into the LSTM cell as input. It is used in conjunction with the input vector (x_t) to compute the candidate cell state ($\tilde{C}_t$) and the gate values (f_t, i_t, and o_t).

8. h_t: Updated hidden state - This is the output of the LSTM cell, which is passed to the next layer or the next time step. It is a combination of the updated cell state (C_t) and the output gate (o_t), allowing the LSTM to control the information passed on to the subsequent layer or time step.

Let's go through the LSTM cell step by step for our stock price vector x.

Forget gate (f_t) calculation: We calculate the forget gate values based on the previous hidden state h_{t-1}, the input vector x_t, and the forget gate weights W_f and bias b_f. For the first step, we can assume h_{t-1} is initialized to zeros.

$$f_t = \sigma(W_f x_t + b_f) = \sigma \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.6 \end{pmatrix} = \sigma \begin{pmatrix} 21 \\ 42 \end{pmatrix}$$

$$i_t = \sigma(W_i x_t + b_i) = \sigma \begin{pmatrix} 0.7 & 0.8 \\ 0.9 & 1.0 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 1.1 \\ 1.2 \end{pmatrix} = \sigma \begin{pmatrix} 84.1 \\ 105.2 \end{pmatrix}$$

$$o_t = \sigma(W_o x_t + b_o) = \sigma \begin{pmatrix} 1.3 & 1.4 \\ 1.5 & 1.6 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 1.7 \\ 1.8 \end{pmatrix} = \sigma \begin{pmatrix} 147.2 \\ 168.3 \end{pmatrix}$$

$$\tilde{C}_t = \tanh(W_C x_t + b_C) = \tanh \begin{pmatrix} 1.9 & 2.0 \\ 2.1 & 2.2 \end{pmatrix} \begin{pmatrix} 100 \\ 105 \end{pmatrix} + \begin{pmatrix} 2.3 \\ 2.4 \end{pmatrix} = \tanh \begin{pmatrix} 210.3 \\ 231.4 \end{pmatrix}$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t = f_t \odot 0 + i_t \odot \tilde{C}_t = i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

The sigmoid function is used in the forget, input, and output gates because its output ranges from 0 to 1, which allows the gates to control the flow of information in a proportional manner. When the sigmoid function's output is close to 0, it blocks information flow, and when it's close to 1, it allows information to flow.

The tanh function is used for the candidate cell state and the final hidden state update because its output ranges from -1 to 1, providing a normalized representation of the internal values.

Now, let's assume the actual stock price for the next day is $y = 112$. We can calculate the loss using the Mean Squared Error (MSE) between the predicted value ($\hat{y} = h_t$) and the actual value (y): In a typical sequence prediction problem using an LSTM, the input data is a sequence of length T, and the model generates a sequence of T predictions. In such a case, N in the MSE loss function would be equal to the number of predictions made by the LSTM, which is T.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

In our stock LSTM algorithm, we a predicting the closing price for the next day based off the previous 60 days We then calculate the loss using the output from that, and the actual closing price In our case, we are predicting a single number, so the equation would be

$$\text{MSE} = (y_i - \hat{y}_i)^2$$

To update the weights and biases, we compute the gradients and adjust the weights using stochastic gradient descent:

$$\frac{\partial \text{MSE}}{\partial W_f} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_f} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_i} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_i} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_o} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_o} = \text{some value}$$

$$\frac{\partial \text{MSE}}{\partial W_C} = \text{some value}, \frac{\partial \text{MSE}}{\partial b_C} = \text{some value}$$

In practice, the gradient calculations involve more complex partial derivatives, and we would use a deep learning framework to compute them automatically using backpropagation through time.

Using the computed gradients, we update the weights and biases with stochastic gradient descent:

$$W_f = W_f - \alpha \frac{\partial \text{MSE}}{\partial W_f}, b_f = b_f - \alpha \frac{\partial \text{MSE}}{\partial b_f}$$

$$W_i = W_i - \alpha \frac{\partial \text{MSE}}{\partial W_i}, b_i = b_i - \alpha \frac{\partial \text{MSE}}{\partial b_i}$$

$$W_o = W_o - \alpha \frac{\partial \text{MSE}}{\partial W_o}, b_o = b_o - \alpha \frac{\partial \text{MSE}}{\partial b_o}$$

$$W_C = W_C - \alpha \frac{\partial \text{MSE}}{\partial W_C}, b_C = b_C - \alpha \frac{\partial \text{MSE}}{\partial b_C}$$

Where α is the learning rate.

After updating the weights and biases, the LSTM would go through the sequence again, and the process would be repeated multiple times (epochs) until the loss converges to a minimum value.