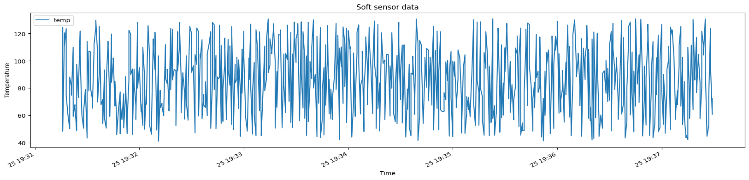
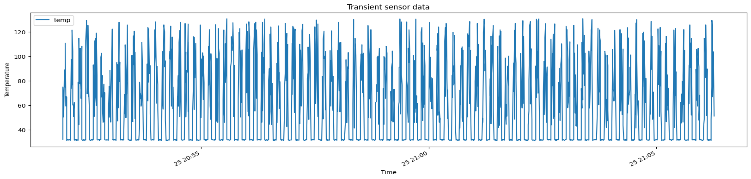
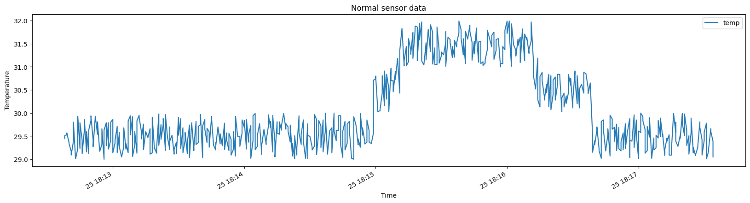
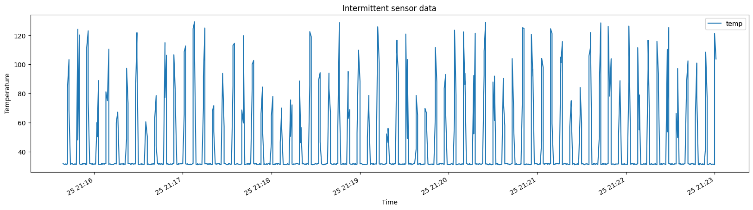
**Temperature Sensor Fault Detection**

About data :

* four types of sensor data : normal, intermitted fault, soft fault, transient fault.
* Here data is sequence data because it is collected over time, and recorded in a specific order.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Time duration | Number of records |
| 0 | Normal data | 18:12:38 to 18:17:34 (5 min) | 566 |
| 1 | Intermitted data | 21:15:39 to 21:23:01 (7 min) | 879 |
| 2 | Soft data | 19:31:16 to 19:37:29 (6 min) | 711 |
| 3 | Transient data | 20:51:58 to 21:06:16 (14 min) | 1709 |

Total = 3865



After plotting each type of data, We can see each follows some pattern.

**LSTM( long short term memory) :**

LSTM is excellent at handling data that has a time-based or sequential structure. It is a variant of RNN to overcome limitations of long term pattern learning.

LSTM has ability to remember and learn from past information. It stores important information for long time and forget less important information.

LSTM uses gates to control the flow of information through the network.

* Input gate: The input gate controls how much of the current input is added to the cell state.
* Forget gate: The forget gate controls how much of the previous cell state is forgotten.
* Output gate: The output gate controls how much of the current cell state is output from the LSTM cell.
* Cell state: The cell state is a long-term memory that stores the current state of the LSTM cell.

At each time step, the LSTM cell takes the current input and the previous hidden state as input and outputs the current hidden state. The current hidden state is then passed to the next LSTM cell in the sequence.

* **Accuracy using LSTM : 0.9947**

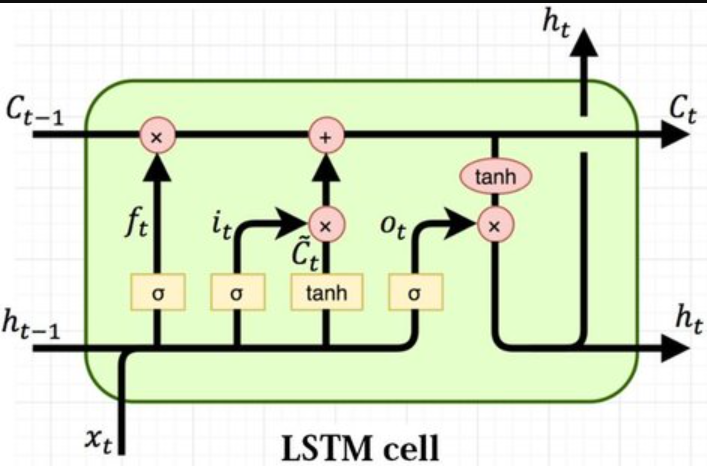
**Approach :**

1. Reading sensor data from multiple excel files, each represents type of sensor data (normal, intermitted, soft, transient).
2. Plotted sensor data to visualize variations over time for each type.
3. Assigned labels to each type of data (0,1,2,3) and combined them.
4. Transformed Data into a time series format, each sequence contains 50 consecutive data points.
5. Divided data into train test sets, 70% for training, 30% for testing.
6. Created a sequential LSTM model with 2 layers for data processing and a dense layer with a ‘softmax’ activation function for multi-class classification.
7. Compiled model with ‘adam’ optimizer and sparse categorical cross-entropy loss function.
8. Trained for 15 epochs, with batch size of 64 and 20% validation split.

|  |  |  |
| --- | --- | --- |
| **Number of Faulty Records** | **Precision (Faulty Records)** | **False Rate (Faulty Records)** |
| 330 | 0.995098 | 0.004902 |
| 656 | 0.993276 | 0.006724 |
| 984 | 0.992488 | 0.007512 |
| 1326 | 0.992122 | 0.007878 |
| 1664 | 0.991149 | 0.008851 |

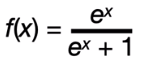
**LSTM steps for initial unit:**

Each LSTM unit will process one entire sequence of size 50 records. Here 3433 sequences for 64 units.



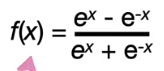
**Sigmoid activation function:**

Takes any x- axis coordinate and turns into a y-axis coordinate between 0 and 1. Using below equation.

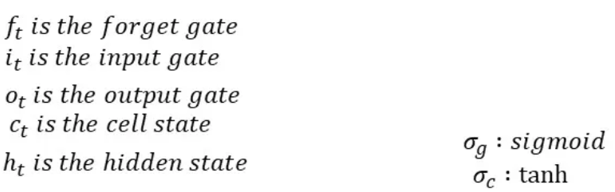


**Tanh activation function (hyperbolic tangent) :**

Takes any x-axis coordinate and turns into y-axis coordinate between -1 and 1. Using below equation.



* Initially cell state and hidden state =0
* Initially weights and biases are generated randomly and gradually get adjusted to minimize the loss function.



F­t­ = forget gate

I­t ­ = input gate

O­t ­ = output gate

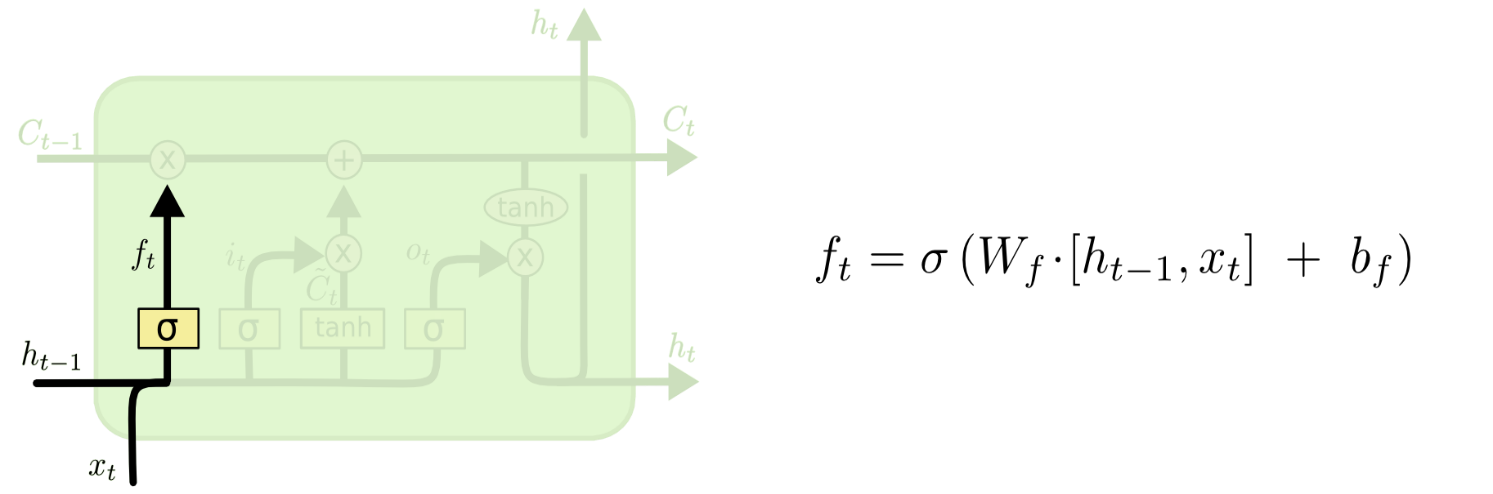
C­­t­ = cell state

Ht = hidden state

**W = -0.11120389, U = -0.2553228, B = 0.020239927, x = 45.54**

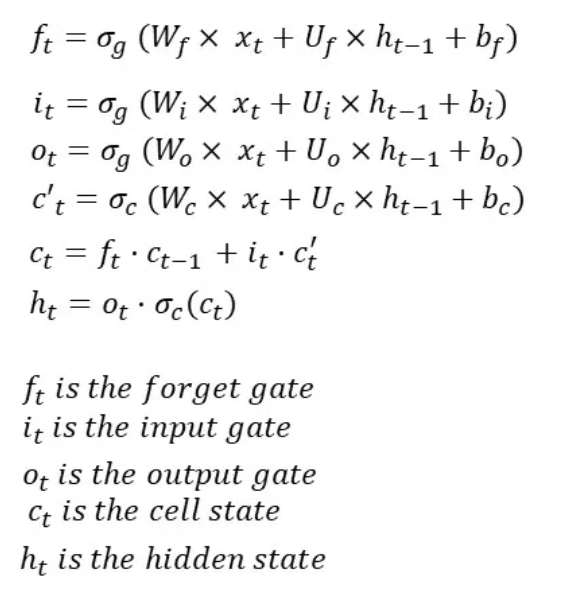
( Weights W,U and bias B taken using lstm\_layer.get\_weights() )

1. What information We are going to throw away from the cell state.



Forget gate layer:

Using ht-1 and xt , this gate outputs a number between 0 and 1. 1 represents “completely keep it” and 0 represents “completely get rid of it”



= sigmoid (-0.11120389 (45.54) + -0.2553228(0) + 0.020239927)

= sigmoid (-5.06422515 + 0.020239927)

= sigmoid (-5.04398522)

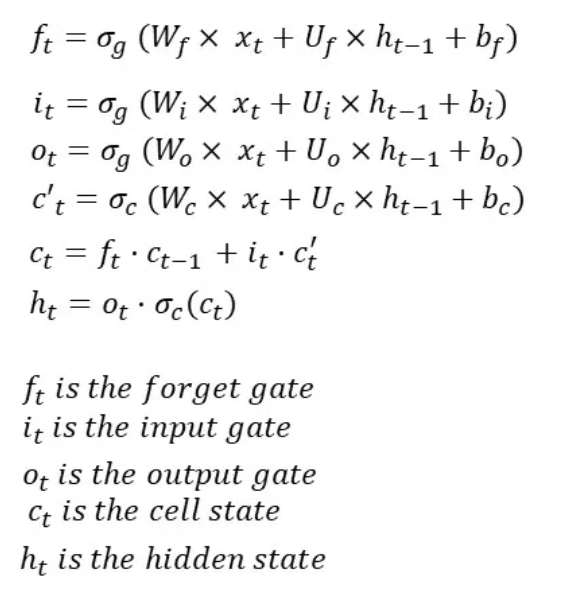
= 0.00640669

1. What new information We are going to store in the cell state.

Input gate layer: decide which value we’ll update.

Tanh layer: creates a vector of new candidate values.



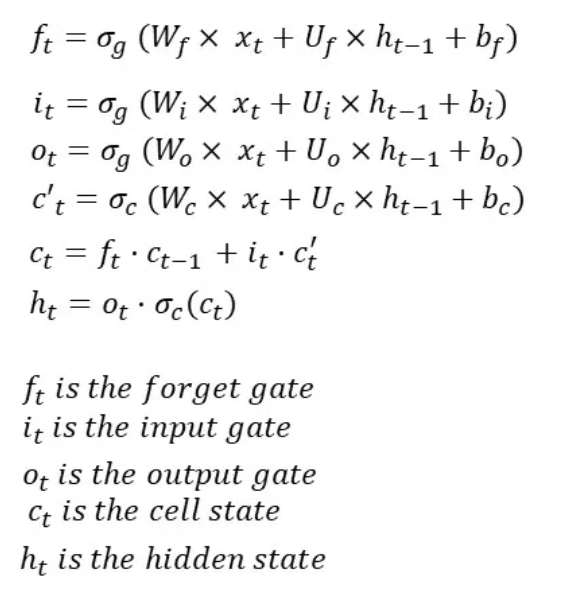


= sigmoid (-0.11120389 (45.54) + -0.2553228(0) + 0.020239927)

= sigmoid (-5.06422515 + 0.020239927)

= sigmoid (-5.04398522)

= 0.00640669



= tanh (-0.11120389 (45.54) + -0.2553228(0) + 0.020239927)

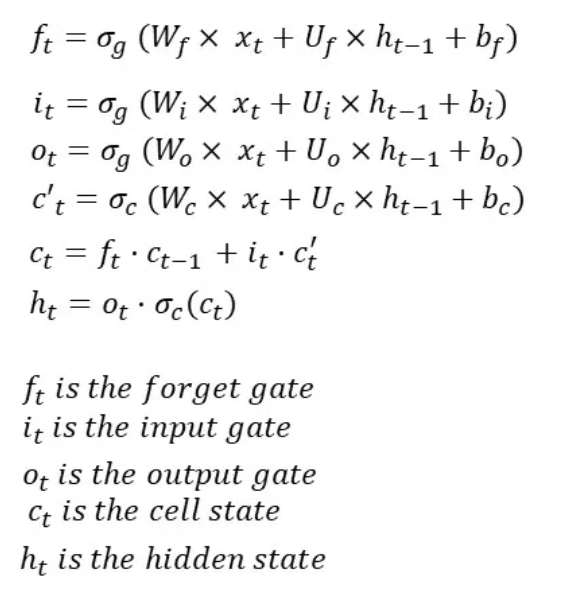
= tanh (-5.06422515 + 0.020239927)

= tanh (-5.04398522)

= -0.9999168

1. Update old cell state Ct-1 to Ct.



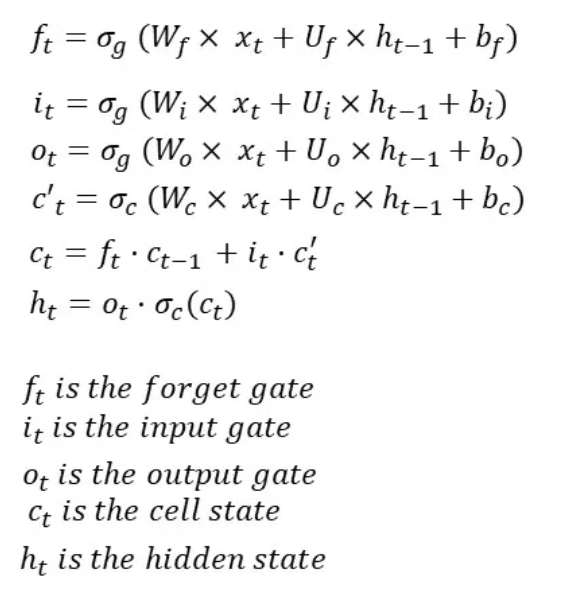


= (0.00640669\* 0) + (0.00640669) \* ( -0.9999168)

= -0.00640615696

1. What we are going to output, based on cell state.



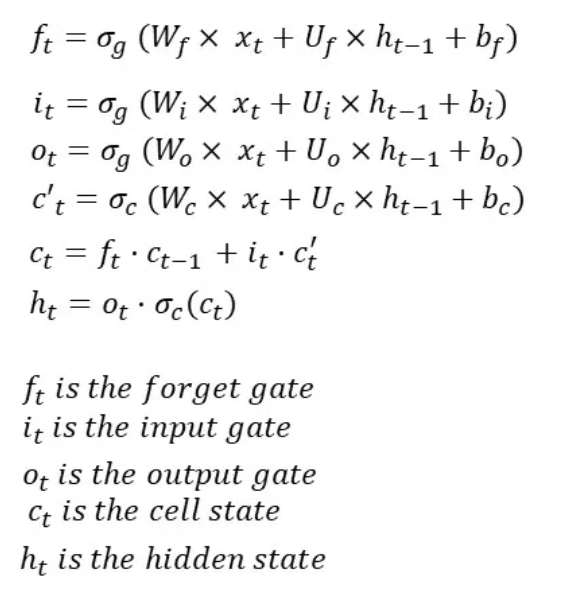


= sigmoid (-0.11120389 (45.54) + -0.2553228(0) + 0.020239927)

= sigmoid (-5.06422515 + 0.020239927)

= sigmoid (-5.04398522)

= 0.00640669



= 0.00640669 \* tanh(-0.00640615696)

= 0.00640669 \* -0.0064060

= -0.00004104125

**DENSE LAYER: (**activation function: softmax)

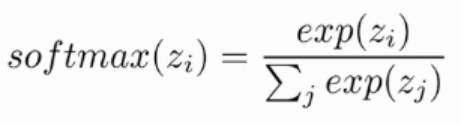
The softmax function is used to produce a probability distribution over multiple classes, making it suitable for multi-class classification tasks.

1. Linear Transformation: Just like any other Dense layer, the input values from the previous layer are subjected to a linear transformation. The key components include multiplying the input values by a set of weights and adding a bias term:

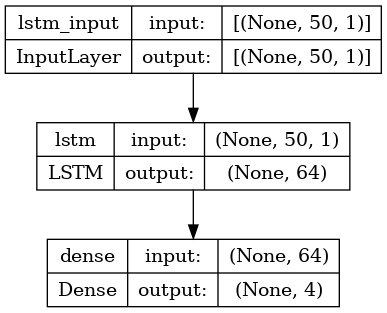
linear\_output = sum(weight\_i \* input\_i) + bias

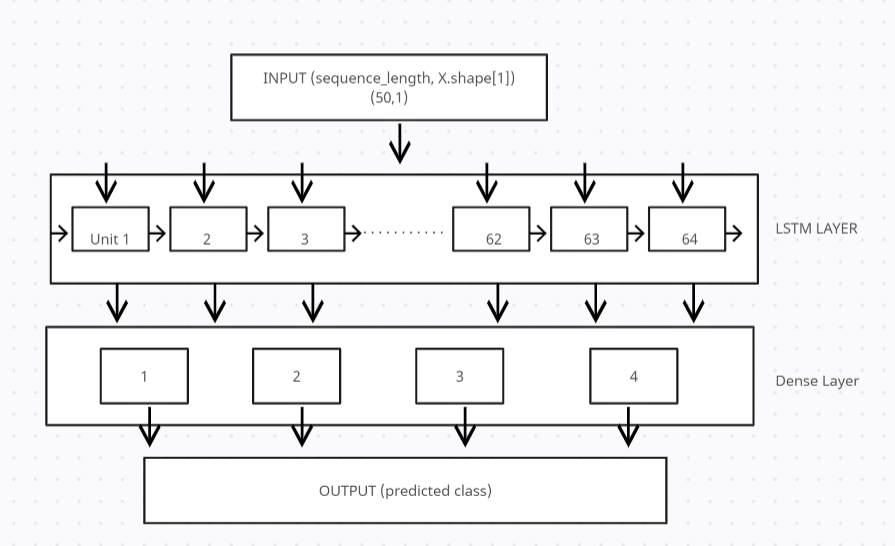
The weights and biases are learnable parameters adjusted during training to optimize the network's predictive ability.

2. Softmax Activation: After the linear transformation, the softmax activation function is applied to the linear output. The softmax function computes the exponentials of the linear output and normalizes them to produce a probability distribution.



Z represents the values from linear transformation output.

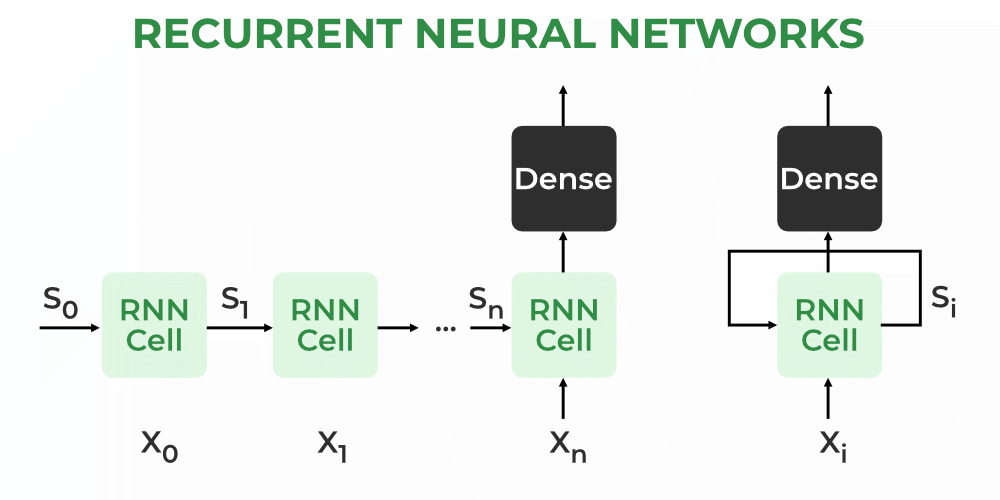




**RNN (Recurrent neural network) :**

Used for sequential and time series data.

Key feature of RNN is to maintain hidden state or memory that stores information about previous time steps In the sequence. RNN has difficulties to learn long range patterns.



* initialized with a hidden state of 0.
* The first element in the sequence is input to the RNN.
* The RNN updates its hidden state based on the input and its previous hidden state.
* The output of the RNN is calculated based on the current hidden state.
* Steps 3 and 4 are repeated for each element in the sequence.
* **Accuracy using RNN : 0.9912**