Monday, October 24, 2022 11:39 PM

1. Problem Description

Four vibration sensors attached at different locations of a machine is collecting information consecutively. The machine is run in two states: **Healthy** and **Faulty** state. The task for us is to create a model that can classify continuous data from the four sensors of the machine, whether the machine is in Healthy or Faulty state.

2. Data

There are two classes of data available in separate csv files.

- Healthy Dataset: 99840 continuous records
- Faulty Dataset: 94208 continuous records
- Total: 99840 + 94208 = 194048 records

I. Train and Test Data

We use 80:20 proportion for Train and Test data.

- **Train Data**:
    - Total: 155239 records
    - Healthy: 79872 records
    - Faulty: 75367 records

    - Row 1 --> Row 79872 : Healthy continuous records
    - Row 79873 --> Row 155239: Faulty continuous records

- **Test Data:**
    - Total: 38809 records
    - Healthy: 19968 records
    - Faulty: 18841 records

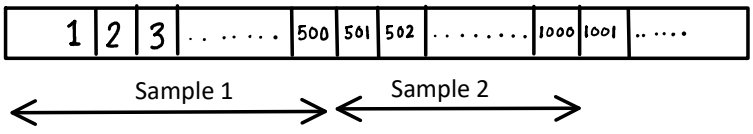
    - Row 1 --> Row 19968 : Healthy continuous records
    - Row 19969 --> Row 38809: Faulty continuous records

II. Sample Selection

- **Sample Size**: For training purpose, we use variable training sample size(500-1000) for training our model.

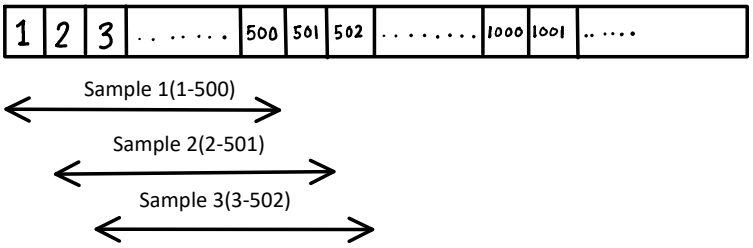
- **Selection of sample from continuous data:**

    - Selecting samples which do not have overlapping data.



- In this way, for a train record length of 155239 and minimum sample size of 500, the maximum number of samples we can get is : floor(155239/500) = 310 samples
- **Issue:** The number of samples present for training seems to be low.

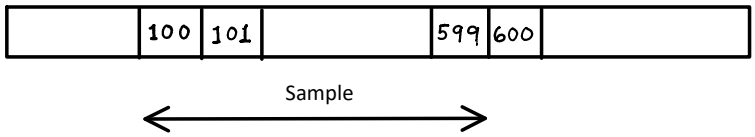
○ Selecting overlapping samples



- In this way, for a train record length of 155239 and minimum sample size of 500, the maximum number of samples we can get is: 155239-500 = 154739 samples
- This is a good number of samples to train on.
- We use this method for training our model

- **<u>Sample starting point</u>**: To select a continuous data sample from the dataset we only need the starting index of the dataset.

○ Eg. For a sample size : 500, starting index:100
- dataset[(starting index): (starting index + sample size)]
- This gives data from Row-100 to Row-599



○ For our training data, given the following
- Healthy continuous records: Row 1 --> Row 79872
- Faulty continuous records: Row 79873 --> Row 155239
- Sample size(say): 500

The starting index is restricted to : [1 - (79872-500) , 79873-(155239-500)]
=> [1 - 79372 , 79873-154739]

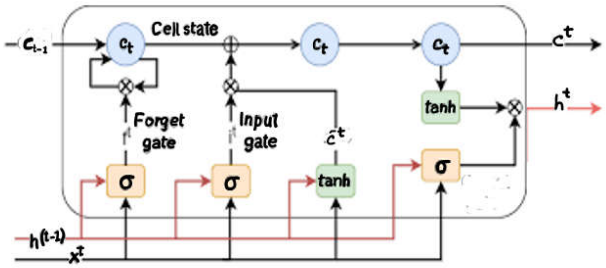
- We create a list, containing starting index value of the aforesaid range. We use this to get a sample from the training data.

○ **Note:** Selecting staring index value between [79373 - 79872], will result in a sample mixed healthy and faulty data. Hence the index value cannot have continuous value from 1-154793.

3. <u>Model</u>

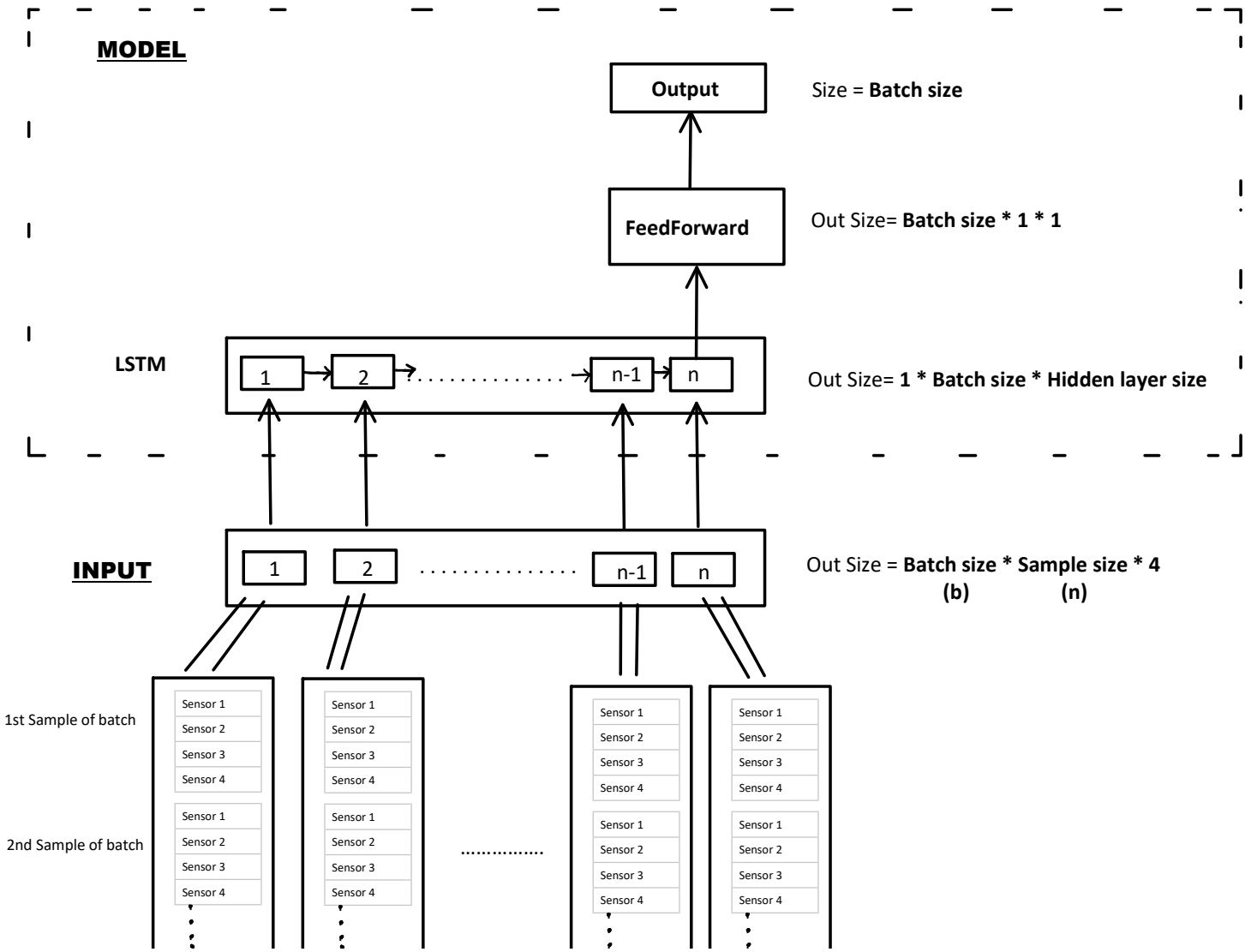
Since our aim is to predict from a continuous set of samples, which are in sequence, therefore we will be using **Recurrent Neural Networks(RNN)** for our model.

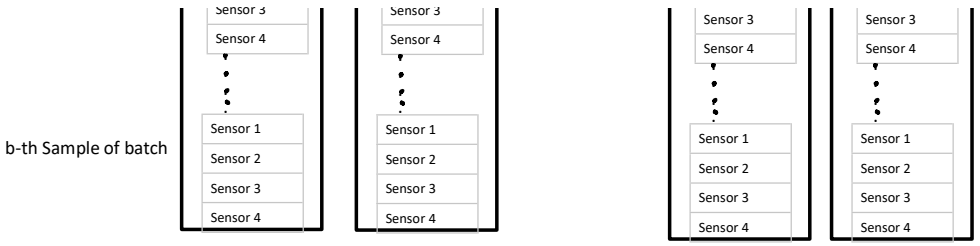
A typical RNN performs poorly for longer sequence, due to vanishing gradient problem. Thus, we use **Long Short Term Memory(LSTM),** a special kind of RNN, which handles long sequences well. Hence, we use LSTM in our model. Diagram of a typical LSTM block is given below.



A typical LSTM Block

For our model, we will have one LSTM layer followed by a Feedforward Layer. The Below Diagram illustrates:





Input Data

- Each row of the sample of 4 sensor values.
- Each Sample of size 'n'(sample size) has 'n' number of continuous rows of data.
- Each batch of size 'b'(batch size) has 'b' random samples from the entire dataset.
- The selection of the random samples comes from a list, that has the starting positions of b samples in the batch.

Forward Propagation

- Each batch of samples is fed to the LSTM layer as input.
- The hidden layer of LSTM is computed with input data. The output from the final layer of LSTM is passed to the FeedForward network as input.
- The Feedforward network computes the output for all the batches

Model Parameters for initialization

- **Input Size**: Input size refers to the number of column in each row of data. The size is **4**, coming from 4 sensor data in each row.
- **Hidden Size:** This refers to the size of each block of LSTM.
- **Output Size:** The size of output going out from the LSTM.

4. Model Training

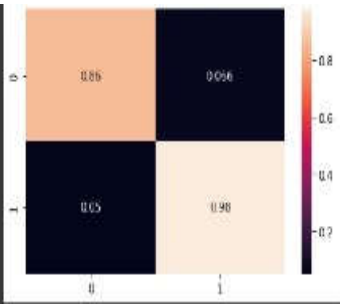
Hyperparameters Tuning:

- **Hidden layer size :** Large hidden layer size increases the ability to fit a given data. However, the computation and time taken is proportional to the size. Our object is to predict the model under 10 milliseconds. Thus we need to use a relatively low value. The optimum value is found to be 32.

- **Batch size:** Since the Hidden layer size is low, higher batch size helps in regularization. However, it increases the training time for the network model. The optimum value is found to be 512.

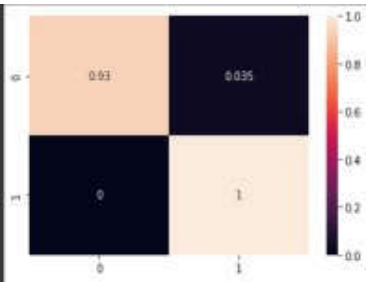
    - Below is the case where model was trained with batch size 16 in 3000 iterations, where the loss was reduced below 0.05. However, the accuracy on the test set is lower.

	precision	recall	f1-score	support
Faulty	0.86	0.94	0.90	288
Healthy	0.98	0.94	0.96	712
accuracy			0.94	1000
macro avg	0.92	0.94	0.93	1000
weighted avg	0.94	0.94	0.94	1000

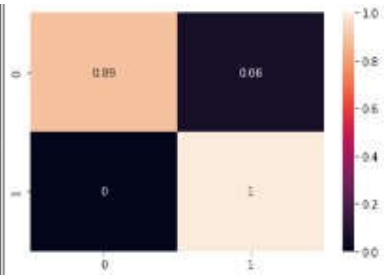


- **Number of LSTM layers:** Again we cannot afford to have multiple LSTM layers due to model prediction time limit. Hence only one layer is used.
- **Learning rate:** Due to small hidden layer size, the number of parameters is less for the network. So, we avoid larger learning rate. The optimum value is found to be 0.0001.
- **Loss Function:** Binary Cross Entropy Loss works well for classification tasks, hence we use it in our model.
- **Optimizer:** Similar to the loss function, Adam optimizer works well in our case.
- **Sample size:** In the problem statement, the sample size is advised from 500-1000. However, if train our model with only one fixed sample size(between 500-1000). Then on testing if the sample size is changed then our model gives low accuracy.
    - Below is the case where the model was trained with 500 sample size. The testing accuracy for 500 sample size was good but for 1000 sample size, the accuracy went down a bit.

	precision	recall	f1-score	support
Faulty	0.93	1.00	0.97	327
Healthy	1.00	0.97	0.98	673
accuracy			0.98	1000
macro avg	0.97	0.98	0.97	1000
weighted avg	0.98	0.98	0.98	1000



	precision	recall	f1-score	support
Faulty	0.89	1.00	0.94	311
Healthy	1.00	0.94	0.97	689
accuracy			0.96	1000
macro avg	0.94	0.97	0.96	1000
weighted avg	0.97	0.96	0.96	1000



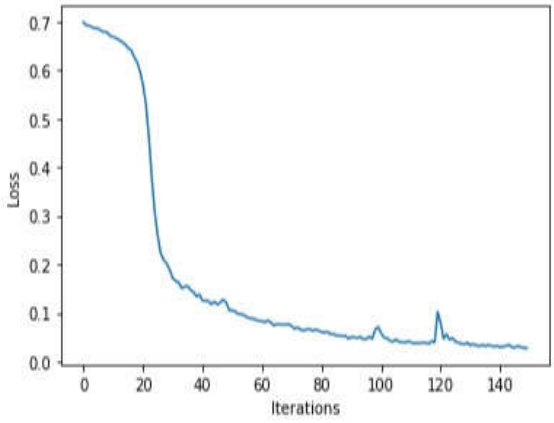
To solve this issue we train a model with varying sample size between 500-1000. This resolves the issue a bit.

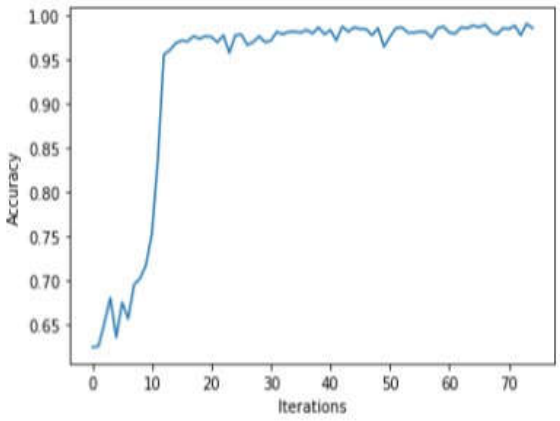
Training Loss and Accuracy

Like discussed above, we find the optimum values for the hyperparameters through training. With our LSTM **hidden size** as **32**, we iterate over **3000 iterations** with **batch size** of **512 samples** with **varying**

sequence length, using **Adam optimizer** with **learning rate** of **0.0001**, we train our model.

- **Model Training time:** 12 minutes 36 seconds
- Below we see the Training Loss and Accuracy through various iterations.

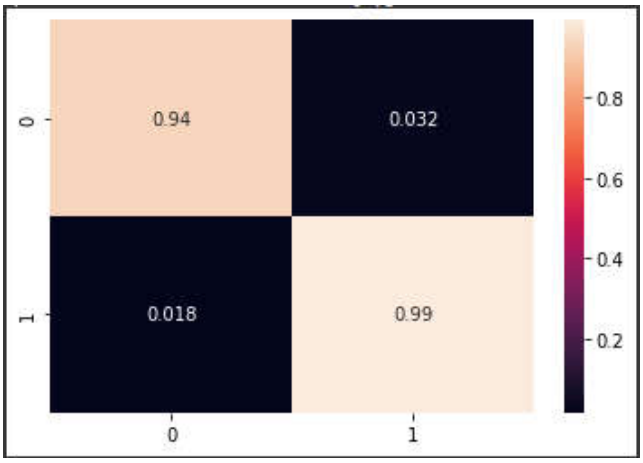




5. Result Analysis

After training our model, we test it's accuracy on the test data. We vary our sample size in testing.

	precision	recall	f1-score	support
Faulty	0.94	0.98	0.96	12177
Healthy	0.99	0.97	0.98	25823
accuracy			0.97	38000
macro avg	0.96	0.97	0.97	38000
weighted avg	0.97	0.97	0.97	38000



Above is the classification report and confusion matrix for the testing.

Observations and Insights:

- The precision for the Faulty state is lower than the Healthy state

        - **Insight:** There is a slight higher tendency of the model to predict Healthy state than Faulty state. This can be due to imbalanced data. Higher supports are present for Healthy state than for Faulty states in both training and testing.

    - The recall for the Healthy and Faulty state are similar.

    - Overall F1 score for Healthy state is more for Faulty state. F1 scores for all states(average, micro average and weighted average) is above 0.95.

    - Overall accuracy of the model comes up around 0.97.

    - The Prediction time for a given sample is under 8 milliseconds.

Further Improvement Ideas

The accuracy of the model can be increased by the following:

- Balancing the dataset: There are more healthy data than faulty data in training. Hence, increasing the faulty data might help.
- Better tuning of the Hyperparameters: There can always be some better tuning of the hyperparameters, given more time!

6. How is the model good for deployment

- The model accuracy is above 97% for the testing data. So it predicts well enough for unknown data, on which it is not trained upon.

- The model takes less than 8 milliseconds to predict the state of the model. Hence, it can be used for real time prediction.

- The model performs well for wide sample size range. This gives added robustness to the model.

- The size of the model is relatively small for deep network. It can be loaded into the memory quite quickly.

- The requirements for the model is just the continuous sample data of length of 500-1000.