B.E CSE VI Q BATCH

CS6301-MACHINE LEARNING

PREDICTIVE MAINTENANCE OF AIRCRAFT ENGINE USING LSTM NETWORKS

TEAM NUMBER : 03

TEAM MEMBERS:

VIGNESH G - 2018103078

DEEKSHITH M -2019103014

AJITESH M - 2019103503

HEMANTH D - 2019103020

PROBLEM STATEMENT

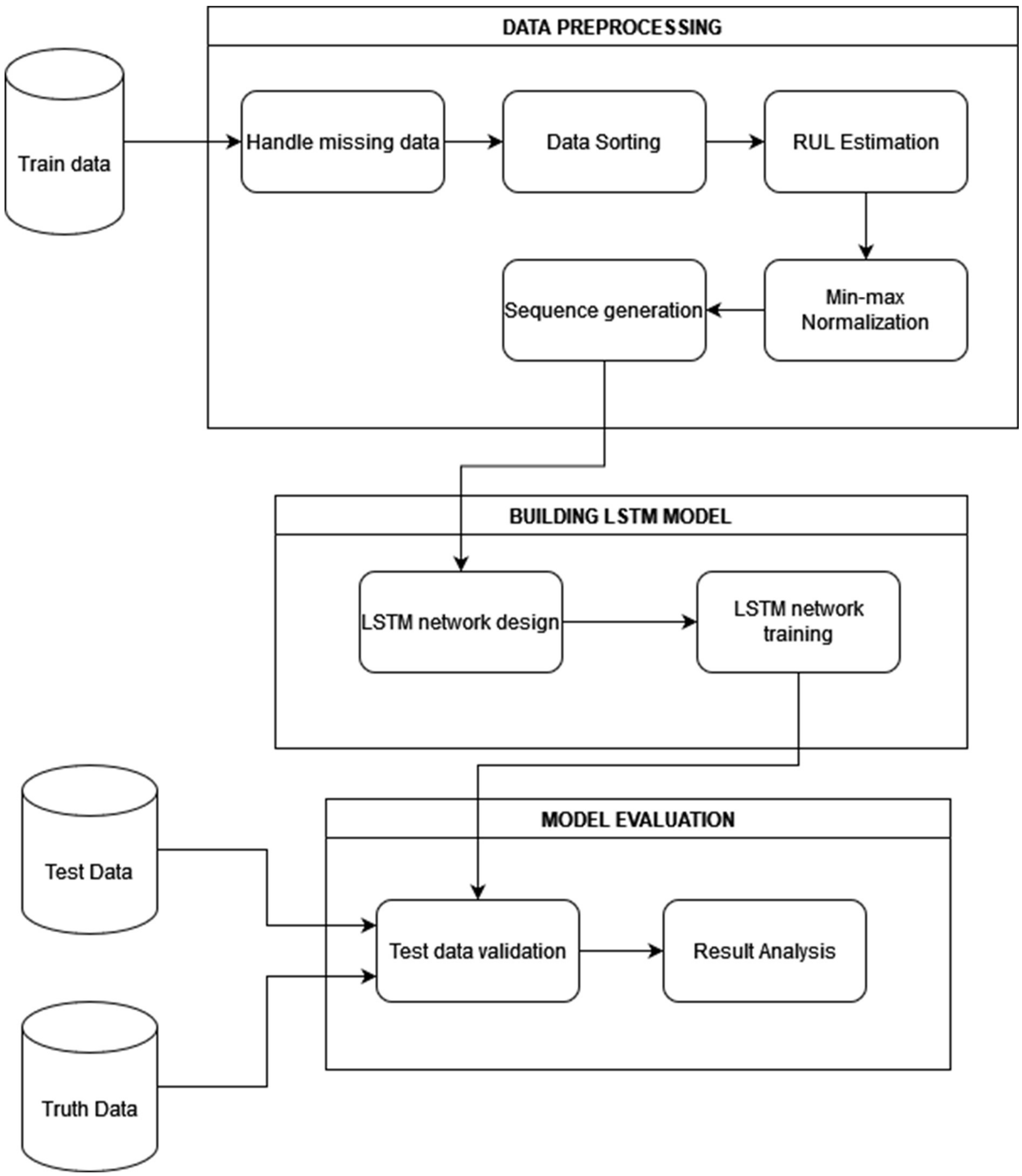
What if a part of aircraft could let know when the aircraft component needed to be replaced or repaired? It can be done with continuous data collection, monitoring and advanced analytics. In the aviation industry, predictive maintenance promises increased reliability as well as improved supply chain and operational performance. The main goal is to ensure that the engines work correctly under all conditions and there is no risk of failure. The main source of data regarding the health of the engine is measured during the real time. Several variables are calculated, including fan speed, quantity and oil pressure and environmental variables such as temperature, air speed.

Aspects related to the maintenance have become especially failure of a have catastrophic consequences. Current systems have the ability to warn inform only when the failure occurred. An early warning system that predicts the occurrence of component failure is required. A machine learning approach, using LSTM networks, is proposed to predict the RUL (Remaining Useful Life) by analysing failure patterns in the past. Training of LSTM networks are carried out on a high performance large-scale processing engine.

OBJECTIVE

A machine learning approach through the use of Long Short Term Memory (LSTM) networks is used to analyse sensor time series sequences to estimate the RUL of turbofan engines, and we provide an analysis where we show how the LSTM performance changes when varying its internal hyperparameters.

OVERALL ARCHITECTURE DIAGRAM



DETAILS OF MODULE DESIGN

1.Data pre-processing

2. Building LSTM network

3. Training the model

4. Testing the model and Result analysis.

1.DATA PRE-PROCESSING:

The dataset has inputs.

- Training data: It is the aircraft engine run-to-failure data.
- Testing data: It is the aircraft engine operating data without failure events recorded.
- Ground truth data: It contains the information of true remaining cycles for each engine in the testing data.

The training data ("train_FD001.txt") consists of multiple multivariate time series with "cycle" as the time unit, together with 21 sensor readings for each cycle. Each time series can be assumed as being generated from a different engine of the same type. Each engine is assumed to start with different degrees of initial wear and manufacturing variation, and this information is unknown to the user. In this simulated data, the engine is assumed to be operating normally at the start of each time series. It starts to degrade at some point during the series of the operating cycles. The degradation progresses and grows in magnitude. When a predefined threshold is reached, then the engine is considered unsafe for further operation. In other words, the last cycle in each time series can be considered as the failure point of the corresponding engine**.** Taking the sample training data shown in the following table as an example, the engine with id=1 fails at cycle 192, and engine with id=2 fails at cycle 287.with this information we can calculate RUL for training dataset

The testing data ("test_FD001.txt") has the same data schema as the training data. The only difference is that the data does not indicate when the failure occurs (in other words, the last time period does NOT represent the failure point). Taking the sample testing data shown in the following table as an example, the engine with id=1 runs from cycle 1 through cycle 31. It is not shown how many more cycles this engine can last before it fails.

The ground truth data ("RUL_FD001.txt") provides the number of remaining working cycles for the engines in the testing data. Taking the sample ground truth data shown in the following table as an example, the engine with id=1 in the testing data can run another 112 cycles before it fails.

2.BUILDING THE MODEL

Long shot term Memory Network Known as LSTM. It is one of the best kind of RNN with capability of avoiding gradient dispersion. It is designed to avoid long term dependencies. LSTM Cells are where data is transfers and updated, cell states of the LSTM is changed as compared to RNN. Network is based on short term states, long term states and its three gates: input gate, output gat and forget gate. Where ft is forget gate use to forget the information that is no longer required.

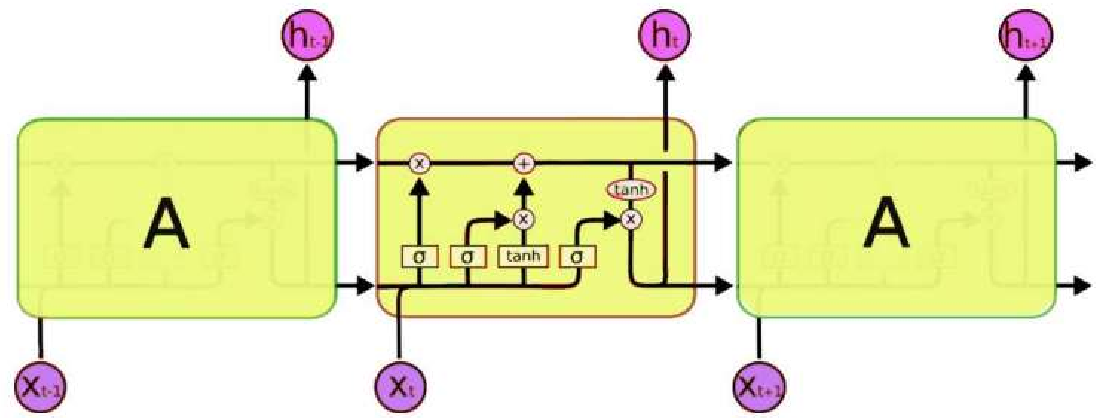


Figure 2: LSTM architecture

$$ft = \sigma(Wf \cdot [ht-1, xt] + bf)$$

here σ is activation function x is input to the gate and b is bias vector.

The input gate contains two path one for the new input and second for vector I generated by forget gate, which is use to modify the cell state.

$$it = \sigma(Wi \cdot [ht-1, xt] + bi)$$

$$C\tilde{}\ t = \tanh(Wc \cdot [ht-1, xt] + bc)$$

Where W are the weight matrices and b are the bias vectors for input gate with activation function of tanh. And updated state of gate is:

$$Ct = ft \otimes Ct-1 + it \otimes C$$

output gates have two parts as input gate.

$$ot = \sigma(Wo \cdot [ht-1, xt] + bo)$$

$$ht = ot \otimes \tanh(Ct)$$

3.TRAINING THE MODEL.

We input our training dataset for which we know the Remaining Useful lifetime[RUL] ,our LSTM model takes this as input and calculates the RUL. From our experience, we noticed that the obtained results can be very different when varying the model architecture so the base idea of such an analysis is to show how the model performance, in terms of Root Mean Squared Error (RMSE), changes over a different set of hyperparameters. RMSE is commonly used in supervised learning applications, as RMSE uses and needs true measurements at each predicted data point.

Root mean square error can be expressed as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \|y(i) - \hat{y}(i)\|^2}{N}},$$

where N is the number of data points, y(i) is the i-th measurement, and ŷ(i) is its corresponding prediction.

We were able to analyze the RMSE variations when changing the hyperparameters values and understand which are the elements that affects it mostly. The result of such an analysis is very important because it allows to have a better understanding on how this kind of models work and how we can improve them by changing their configuration. Finally we fix the hyperparameters which give us high accuracy.

4.TESTING THE MODEL AND RESULT ANALYSIS.

We finally apply the Training dataset whose ground truth Remaining useful lifetime is is given in a separate dataset into out trained and optimized LSTM model , we obtain the output for each case of aircraft engine based on the aircraft id. The Rul predicted and actual ground truth is compared to analyze the result.

Model performance on the test data showing the differences between the predictions made and the ground truth values where the generic i-th difference has been computed as:

di = y^i − yi

In this sense, a value higher than zero means that our model made an optimistic prediction while a value lower than zero means a pessimistic prediction. Nevertheless, the majority of them presents a difference near to zero meaning that the model is able to correctly predict the RUL with a very good level of accuracy. We analyze the accuracy, error rate using various graph plotting packages such as pyplot, seaborn to get a conclusion of how our LSTM model perform for predicting Remaining useful lifetime for Training dataset.

IMPLEMENTATION

MODULE-1:DATA PREPROCESSING

Required packages and libraries needed for building the model are imported.

```python
import keras
import keras.backend as K
from keras.layers.core import Activation
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout, LSTM

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
from sklearn import preprocessing
```

[3]

```python
# Setting seed for reproducibility
np.random.seed(1234)
PYTHONHASHSEED = 0
```

The dataset containing training and testing data are uploaded and the null values are removed in order to avoid false information.

```python
# read training data - It is the aircraft engine run-to-failure data.
# read test data - It is the aircraft engine operating data without failure events recorded.
# read ground truth data - It contains the information of true remaining cycles for each
#  engine in the testing data.
train_df = pd.read_csv('PM_train.txt', sep=" ", header=None)
test_df = pd.read_csv('PM_test.txt', sep=" ", header=None)
truth_df = pd.read_csv('PM_truth.txt', sep=" ", header=None)
```

[5]

```python
# Drop missing data columns(redundant)
train_df.drop(train_df.columns[[26, 27]], axis=1, inplace=True)
test_df.drop(test_df.columns[[26, 27]], axis=1, inplace=True)
truth_df.drop(truth_df.columns[[1]], axis=1, inplace=True)
```

[6]

```python
# Sorting and indicating columns
train_df.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                     's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                     's15', 's16', 's17', 's18', 's19', 's20', 's21']

train_df = train_df.sort_values(['id','cycle'])

test_df.columns = ['id', 'cycle', 'setting1', 'setting2', 'setting3', 's1', 's2', 's3',
                     's4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
                     's15', 's16', 's17', 's18', 's19', 's20', 's21']
```

[7]

```python
train_df
```

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s12	s13	s14	s15	s16	s17	s18	s19	s20	s21
0	1	1	-0.0007	-0.0004	100.0	518.67	641.82	1589.70	1400.60	14.62	...	521.66	2388.02	8138.62	8.4195	0.03	392	2388	100.0	39.06	23.4190
1	1	2	0.0019	-0.0003	100.0	518.67	642.15	1591.82	1403.14	14.62	...	522.28	2388.07	8131.49	8.4318	0.03	392	2388	100.0	39.00	23.4236
2	1	3	-0.0043	0.0003	100.0	518.67	642.35	1587.99	1404.20	14.62	...	522.42	2388.03	8133.23	8.4178	0.03	390	2388	100.0	38.95	23.3442
3	1	4	0.0007	0.0000	100.0	518.67	642.35	1582.79	1401.87	14.62	...	522.86	2388.08	8133.83	8.3682	0.03	392	2388	100.0	38.88	23.3739
4	1	5	-0.0019	-0.0002	100.0	518.67	642.37	1582.85	1406.22	14.62	...	522.19	2388.04	8133.80	8.4294	0.03	393	2388	100.0	38.90	23.4044
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	-0.0004	-0.0003	100.0	518.67	643.49	1597.98	1428.63	14.62	...	519.49	2388.26	8137.60	8.4956	0.03	397	2388	100.0	38.49	22.9735
20627	100	197	-0.0016	-0.0005	100.0	518.67	643.54	1604.50	1433.58	14.62	...	519.68	2388.22	8136.50	8.5139	0.03	395	2388	100.0	38.30	23.1594
20628	100	198	0.0004	0.0000	100.0	518.67	643.42	1602.46	1428.18	14.62	...	520.01	2388.24	8141.05	8.5646	0.03	398	2388	100.0	38.44	22.9333
20629	100	199	-0.0011	0.0003	100.0	518.67	643.23	1605.26	1426.53	14.62	...	519.67	2388.23	8139.29	8.5389	0.03	395	2388	100.0	38.29	23.0640
20630	100	200	-0.0032	-0.0005	100.0	518.67	643.85	1600.38	1432.14	14.62	...	519.30	2388.26	8137.33	8.5036	0.03	396	2388	100.0	38.37	23.0522

20631 rows × 26 columns

Then the Remaining Useful Life is calculated using the number of cycles given in the dataset for training dataset.

```
[8]
# Data Preprocessing - Train data
# Data Labeling - generate column RUL(Remaining Usefull Life or Time to Failure)
rul = pd.DataFrame(train_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
train_df = train_df.merge(rul, on=['id'], how='left')
train_df['RUL'] = train_df['max'] - train_df['cycle']
train_df.drop('max', axis=1, inplace=True)

# MinMax normalization (from 0 to 1)
train_df['cycle_norm'] = train_df['cycle']
cols_normalize = train_df.columns.difference(['id', 'cycle', 'RUL', 'label1', 'label2'])
min_max_scaler = preprocessing.MinMaxScaler()
norm_train_df = pd.DataFrame(min_max_scaler.fit_transform(train_df[cols_normalize]),
                             columns=cols_normalize,
                             index=train_df.index)
join_df = train_df[train_df.columns.difference(cols_normalize)].join(norm_train_df)
train_df = join_df.reindex(columns=train_df.columns)

print(train_df)
```

```
        id  cycle  setting1  setting2  setting3   s1        s2        s3  \
0        1      1  0.459770  0.166667       0.0  0.0  0.183735  0.406802
1        1      2  0.609195  0.250000       0.0  0.0  0.283133  0.453019
2        1      3  0.252874  0.750000       0.0  0.0  0.343373  0.369523
3        1      4  0.540230  0.500000       0.0  0.0  0.343373  0.256159
4        1      5  0.390805  0.333333       0.0  0.0  0.349398  0.257467
...    ...    ...       ...       ...       ...  ...       ...       ...
20626  100    196  0.477011  0.250000       0.0  0.0  0.686747  0.587312
20627  100    197  0.408046  0.083333       0.0  0.0  0.701807  0.729453
20628  100    198  0.522989  0.500000       0.0  0.0  0.665663  0.684979
20629  100    199  0.436782  0.750000       0.0  0.0  0.608434  0.746021
20630  100    200  0.316092  0.083333       0.0  0.0  0.795181  0.639634

             s4   s5  ...       s14       s15  s16       s17  s18  s19  \
0      0.309757  0.0  ...  0.199608  0.363986  0.0  0.333333  0.0  0.0
1      0.352633  0.0  ...  0.162813  0.411312  0.0  0.333333  0.0  0.0
2      0.370527  0.0  ...  0.171793  0.357445  0.0  0.166667  0.0  0.0
3      0.331195  0.0  ...  0.174889  0.166603  0.0  0.333333  0.0  0.0
4      0.404625  0.0  ...  0.174734  0.402078  0.0  0.416667  0.0  0.0
...         ...  ...  ...       ...       ...  ...       ...  ...  ...
20626  0.782917  0.0  ...  0.194344  0.656791  0.0  0.750000  0.0  0.0
20627  0.866475  0.0  ...  0.188668  0.727203  0.0  0.583333  0.0  0.0
20628  0.775321  0.0  ...  0.212148  0.922278  0.0  0.833333  0.0  0.0
20629  0.747468  0.0  ...  0.203065  0.823394  0.0  0.583333  0.0  0.0
20630  0.842167  0.0  ...  0.192951  0.687572  0.0  0.666667  0.0  0.0

            s20       s21  RUL  cycle_norm
0      0.713178  0.724662  191    0.000000
1      0.666667  0.731014  190    0.002770
2      0.627907  0.621375  189    0.005540
3      0.573643  0.662386  188    0.008310
4      0.589147  0.704502  187    0.011080
...         ...       ...  ...         ...
20626  0.271318  0.109500    4    0.540166
20627  0.124031  0.366197    3    0.542936
20628  0.232558  0.053991    2    0.545706
20629  0.116279  0.234466    1    0.548476
20630  0.178295  0.218172    0    0.551247

[20631 rows x 28 columns]
```

Data values are normalized in order to make data uniform across various engines with different RUL range.

```
# Data Preprocessing - Test data
# MinMax normalization (from 0 to 1)
test_df['cycle_norm'] = test_df['cycle']
norm_test_df = pd.DataFrame(min_max_scaler.transform(test_df[cols_normalize]),
                            columns=cols_normalize,
                            index=test_df.index)
test_join_df = test_df[test_df.columns.difference(cols_normalize)].join(norm_test_df)
test_df = test_join_df.reindex(columns=test_df.columns)
test_df = test_df.reset_index(drop=True)

# We use the ground truth dataset to generate labels for the test data.
# generate column max for test data
rul = pd.DataFrame(test_df.groupby('id')['cycle'].max()).reset_index()
rul.columns = ['id', 'max']
truth_df.columns = ['more']
truth_df['id'] = truth_df.index + 1
truth_df['max'] = rul['max'] + truth_df['more']
truth_df.drop('more', axis=1, inplace=True)

# generate RUL for test data
test_df = test_df.merge(truth_df, on=['id'], how='left')
test_df['RUL'] = test_df['max'] - test_df['cycle']
test_df.drop('max', axis=1, inplace=True)

print(test_df)
```

```
        id  cycle  setting1  setting2  setting3   s1        s2        s3  \
0        1      1  0.632184  0.750000       0.0  0.0  0.545181  0.310661
1        1      2  0.344828  0.250000       0.0  0.0  0.150602  0.379551
2        1      3  0.517241  0.583333       0.0  0.0  0.376506  0.346632
3        1      4  0.741379  0.500000       0.0  0.0  0.370482  0.285154
4        1      5  0.580460  0.500000       0.0  0.0  0.391566  0.352082
...    ...    ...       ...       ...       ...  ...       ...       ...
13091  100    194  0.781609  0.500000       0.0  0.0  0.611446  0.619359
13092  100    195  0.436782  0.416667       0.0  0.0  0.605422  0.537388
13093  100    196  0.465517  0.250000       0.0  0.0  0.671687  0.482014
13094  100    197  0.281609  0.583333       0.0  0.0  0.617470  0.522128
13095  100    198  0.574713  0.750000       0.0  0.0  0.524096  0.666667

             s4   s5  ...       s14       s15  s16       s17  s18  s19  \
0      0.269413  0.0  ...  0.132160  0.308965  0.0  0.333333  0.0  0.0
1      0.222316  0.0  ...  0.204768  0.213159  0.0  0.416667  0.0  0.0
2      0.322248  0.0  ...  0.155640  0.458638  0.0  0.416667  0.0  0.0
3      0.408001  0.0  ...  0.170090  0.257022  0.0  0.250000  0.0  0.0
4      0.332039  0.0  ...  0.152751  0.300885  0.0  0.166667  0.0  0.0
...         ...  ...  ...       ...       ...  ...       ...  ...  ...
13091  0.566172  0.0  ...  0.584890  0.564063  0.0  0.500000  0.0  0.0
13092  0.671843  0.0  ...  0.572350  0.485956  0.0  0.583333  0.0  0.0
13093  0.414754  0.0  ...  0.605326  0.507888  0.0  0.583333  0.0  0.0
13094  0.626435  0.0  ...  0.622046  0.562524  0.0  0.583333  0.0  0.0
13095  0.721472  0.0  ...  0.591908  0.636399  0.0  0.666667  0.0  0.0

            s20       s21  cycle_norm  RUL
0      0.558140  0.661834    0.000000  142
1      0.682171  0.686827    0.002770  141
2      0.728682  0.721348    0.005540  140
3      0.666667  0.662110    0.008310  139
4      0.658915  0.716377    0.011080  138
...         ...       ...         ...  ...
13091  0.395349  0.418669    0.534626   24
13092  0.333333  0.528721    0.537396   23
13093  0.372093  0.429301    0.540166   22
13094  0.403101  0.518779    0.542936   21
13095  0.434109  0.402237    0.545706   20

[13096 rows x 28 columns]
```

Data is reshaped into a sequence of length 60. Each sequence contains samples, time steps, features. Engine whose values are less than sequence length are omitted, because it cannot be used for prediction. LSTM model takes sequence of input and find RUL for first value of next sequence.

```python
# Window size extension to 60
sequence_length = 60

# function to reshape features into (samples, time steps, features)
def gen_sequence(id_df, seq_length, seq_cols):
    data_matrix = id_df[seq_cols].values
    num_elements = data_matrix.shape[0]
    for start, stop in zip(range(0, num_elements - seq_length), range(seq_length, num_elements)):
        yield data_matrix[start:stop, :]


# pick the feature columns
sensor_cols = ['s' + str(i) for i in range(1, 22)]
sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm']
sequence_cols.extend(sensor_cols)

# print(sequence_cols)
# val is a list of 192 - 60 = 142 bi-dimensional array (60 rows x 25 columns)
val = list(gen_sequence(train_df[train_df['id'] == 1], sequence_length, sequence_cols))
print(len(val))

# generator for the sequences
# transform each id of the train dataset in a sequence
seq_gen = (list(gen_sequence(train_df[train_df['id'] == id], sequence_length, sequence_cols))
           for id in train_df['id'].unique())

# generate sequences and convert to numpy array
seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
print(seq_array.shape)
```

```python
# function to generate labels
def gen_labels(id_df, seq_length, label):
    data_matrix = id_df[label].values
    num_elements = data_matrix.shape[0]
    return data_matrix[seq_length:num_elements, :]

# generate labels
label_gen = [gen_labels(train_df[train_df['id'] == id], sequence_length, ['RUL'])
             for id in train_df['id'].unique()]

label_array = np.concatenate(label_gen).astype(np.float32)
print(label_array.shape)
print(label_array[:10])
```

```
(14631, 1)
[[131.]
 [130.]
 [129.]
 [128.]
 [127.]
 [126.]
 [125.]
 [124.]
 [123.]
 [122.]]
```

MODULE-2: BUILDING LSTM MODEL

2 layers of LSTM cells are built, with 100 units in the first layer and 50 units in the second layer. In the final layer, sigmoid activation function is included.

```python
# Modeling
model_path = 'regression_model.h5'

def r2_keras(y_true, y_pred):
    SS_res = K.sum(K.square(y_true - y_pred))
    SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
    return (1 - SS_res / (SS_tot + K.epsilon()))

# Network Architecture
# The first layer is an LSTM layer with 100 units followed by another LSTM layer with 60 units.
# Dropout is also applied after each LSTM layer to control overfitting.
# Final layer is a Dense output layer with single unit and linear activation
# since this is a regression problem.
nb_features = seq_array.shape[2]
nb_out = label_array.shape[1]

try:
    f = open(model_path)
    print("Trained model already exists")

except IOError:
    print("Initialize a model")
    model = Sequential()
    model.add(LSTM(
        input_shape=(sequence_length, nb_features),
        units=100,
        return_sequences=True))
    model.add(Dropout(0.3))
    model.add(LSTM(
        units=sequence_length,
        return_sequences=False))
    model.add(Dropout(0.3))
    model.add(Dense(units=nb_out))
    model.add(Activation("linear"))
    model.compile(loss='mean_squared_error', optimizer='rmsprop', metrics=['mae', r2_keras])

    print(model.summary())
```

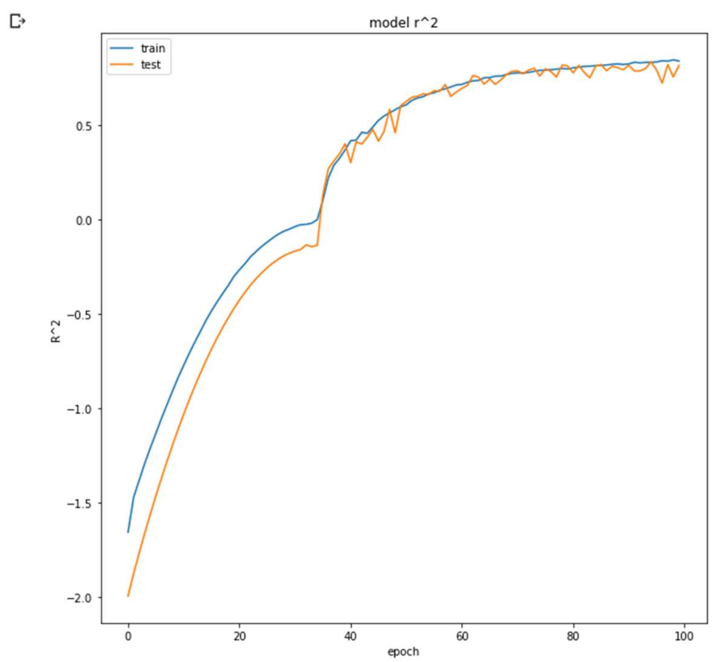
```
Initialize a model
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60, 100)           50400

 dropout (Dropout)           (None, 60, 100)           0

 lstm_1 (LSTM)               (None, 60)                38640

 dropout_1 (Dropout)         (None, 60)                0

 dense (Dense)               (None, 1)                 61

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 89,101
Trainable params: 89,101
Non-trainable params: 0
```

MODULE-3:TRAINING LSTM MODEL

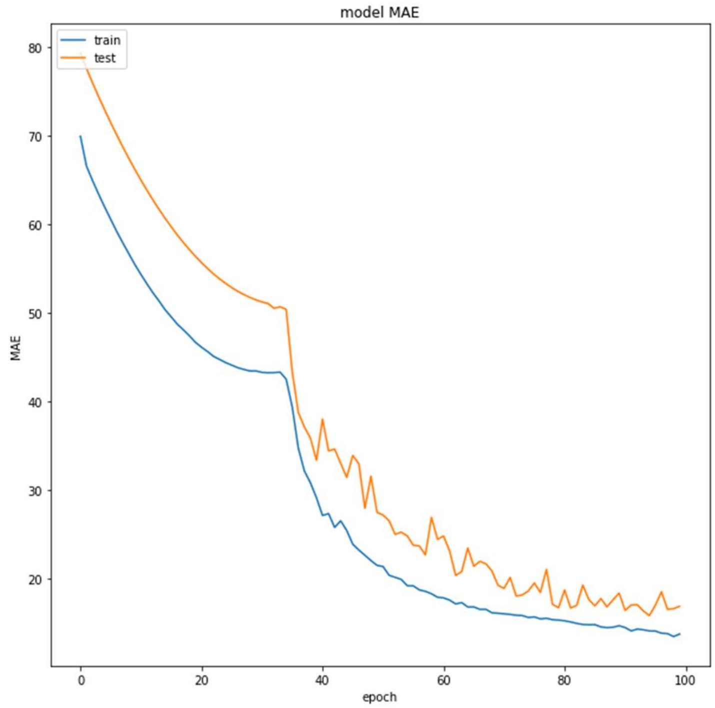
The model takes the dataset as the input and calculates the Remaining Useful Lifetime. The loss is calculated after each epoch.

```
bs = 400
# fit the network
history = model.fit(seq_array, label_array, epochs=100, batch_size=bs, validation_split=0.1, verbose=1,
                    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0,mode='min'),
                               keras.callbacks.ModelCheckpoint(model_path, monitor='val_loss', save_best_only=False,
                                                               mode='min', verbose=0)]
                    )

# list all data in history
print(history.history.keys())
```

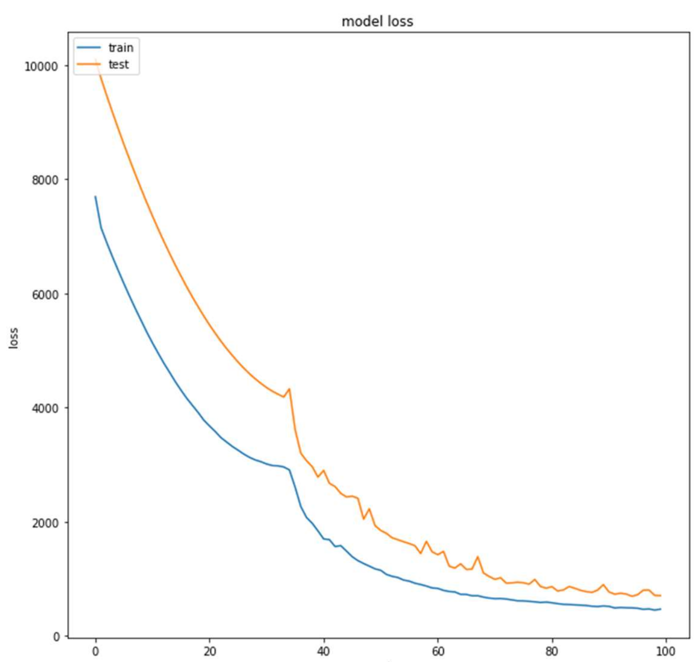
```
# summarize history for R^2
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['r2_keras'])
plt.plot(history.history['val_r2_keras'])
plt.title('model r^2')
plt.ylabel('R^2')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("model_r2.png")
```




```
# summarize history for MAE
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mae'])
plt.plot(history.history['val_mae'])
plt.title('model MAE')
plt.ylabel('MAE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("model_mae.png")
```




```python
# summarize history for Loss
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("model_regression_loss.png")
```



```
# training metrics
scores = model.evaluate(seq_array, label_array, verbose=1, batch_size=bs)
print('\nMAE: {}'.format(scores[1]))
print('\nR^2: {}'.format(scores[2]))

y_pred = model.predict(seq_array, verbose=1, batch_size=bs)
y_true = label_array

test_set = pd.DataFrame(y_pred)
test_set.to_csv('submit_train.csv', index=None)
```

```
37/37 [==============================] - 7s 200ms/step - loss: 434.2534 - mae: 13.1452 - r2_keras: 0.8532

MAE: 13.145186424255371

R^2: 0.8532044291496277
37/37 [==============================] - 8s 199ms/step
```

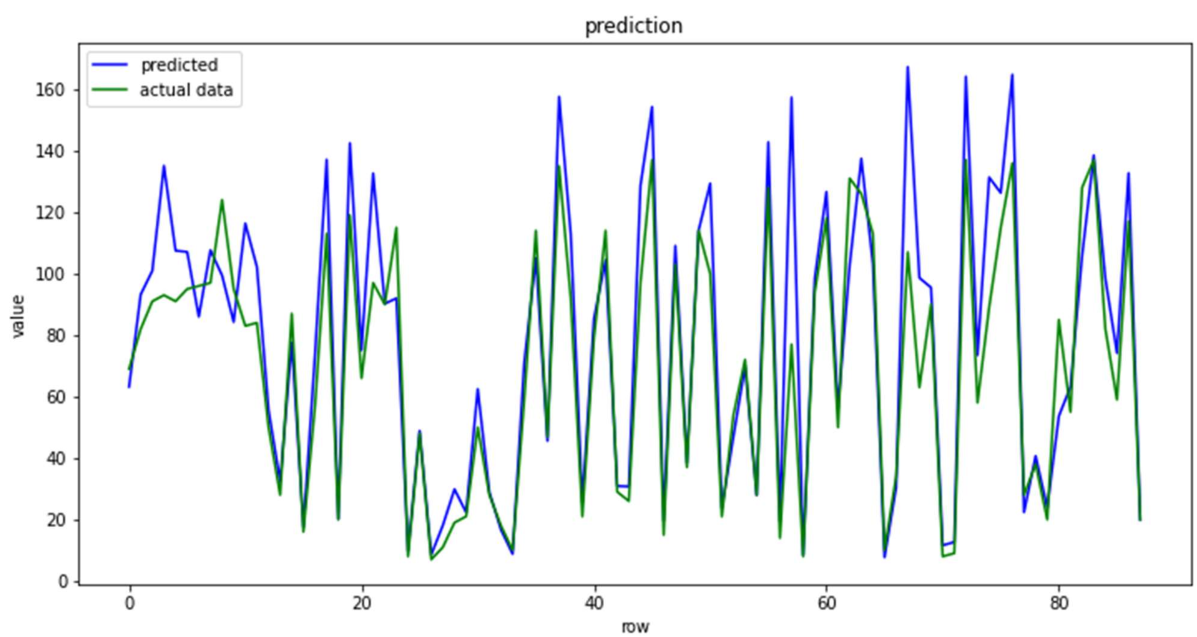
MODULE-4:TESTING THE MODEL

We finally apply the Training dataset whose ground truth Remaining useful lifetime is given in a separate dataset into out trained and optimized LSTM model , we obtain the output for each case of aircraft engine based on the aircraft id. The RUL predicted and actual ground truth is compared to analyze the result.

```
# Test data validation

# We pick the last sequence for each id in the test data
seq_array_test_last = [test_df[test_df['id'] == id][sequence_cols].values[-sequence_length:]
                       for id in test_df['id'].unique() if len(test_df[test_df['id'] == id]) >= sequence_length]

seq_array_test_last = np.asarray(seq_array_test_last).astype(np.float32)
print("seq_array_test_last")
# print(seq_array_test_last)
print(seq_array_test_last.shape)

# Similarly, we pick the labels
# print("y_mask")
y_mask = [len(test_df[test_df['id'] == id]) >= sequence_length for id in test_df['id'].unique()]
label_array_test_last = test_df.groupby('id')['RUL'].nth(-1)[y_mask].values
label_array_test_last = label_array_test_last.reshape(label_array_test_last.shape[0], 1).astype(np.float32)
# print(label_array_test_last.shape)

# if best iteration's model was saved then load and use it
if os.path.isfile(model_path):
    estimator = load_model(model_path, custom_objects={'r2_keras': r2_keras})

    # test metrics
    scores_test = estimator.evaluate(seq_array_test_last, label_array_test_last, verbose=2)
    print('\nMAE: {}'.format(scores_test[1]))
    print('\nR^2: {}'.format(scores_test[2]))

    y_pred_test = estimator.predict(seq_array_test_last)
    y_true_test = label_array_test_last


    pd.set_option('display.max_rows', 1000)
    test_print = pd.DataFrame()
    test_print['y_pred']  = y_pred_test.flatten()
    test_print['y_truth'] = y_true_test.flatten()
    test_print['diff'] = abs(y_pred_test.flatten() - y_true_test.flatten())
    test_print['diff(%)'] = abs(y_pred_test.flatten() - y_true_test.flatten())/y_true_test.flatten()
    print (test_print)
```

```python

test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('submit_test.csv', index=None)

# Plot in blue color the predicted data and in green color the
# actual data to verify visually the accuracy of the model.
fig_verify = plt.figure(figsize=(12, 6))
plt.plot(y_pred_test, color="blue")
plt.plot(y_true_test, color="green")
plt.title('prediction')
plt.ylabel('value')
plt.xlabel('row')
plt.legend(['predicted', 'actual data'], loc='upper left')
plt.show()
fig_verify.savefig("model_regression_verify.png")
```

```
seq_array_test_last
(88, 60, 25)
3/3 - 2s - loss: 353.7944 - mae: 12.7856 - r2_keras: 0.7868 - 2s/epoch - 647ms/step

MAE: 12.7855863571167

R^2: 0.7868390679359436
        y_pred  y_truth       diff   diff(%)
0    63.190376     69.0   5.809624  0.084197
1    93.219536     82.0  11.219536  0.136824
2   100.930458     91.0   9.930458  0.109126
3   135.100723     93.0  42.100723  0.452696
4   107.485336     91.0  16.485336  0.181158
5   107.071800     95.0  12.071800  0.127072
6    86.003746     96.0   9.996254  0.104128
7   107.637825     97.0  10.637825  0.109668
8    99.441872    124.0  24.558128  0.198049
9    84.269844     95.0  10.730156  0.112949
10  116.333488     83.0  33.333488  0.401608
```



PERFORMANCE MEASURES

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n}(\widehat{y_i} - y_i)^2}$$

where y^i and yi are respectively the value predicted by the model and the ground truth provided by the test set of the dataset of the generic i-th input sample, and n is the number of samples of the test set.

REFERENCES

[1] On the use of LSTM networks for Predictive maintenance, in smart industries." - 2019 IEEE International Conference of Smart Computing.

[2] Machine Learning for Equipment failure Prediction and Predictive Maintenance (pm] - shad Griffon - Medium.com

[3] Towards Data Science - Predictive maintenance of turbofan engines - by Keen Peters

[4] Predicting the maintenance of aircraft engines using LSTM - International Journal trend in Scientific research and development. ISSN: 2456-6470

[5] LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras – Ryan – Medium.com

[6] Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) – Brandon Rohrer

[7] Towards Data Science - Illustrated Guide to LSTM’s and GRU’s: A step by step explanation