User manual of the perceptron

Problem Statement

Implement a perceptron to work on the outputs of the test results generated from Thales Alenia Space España test benches during equipment acceptance campaigns by outputting 4 different values namely current_stabilised_value (mA), current_max/min_value (mA), power_state, current_rise/fall_time_spec (mS).

Methodology

- Two neural networks are built one to work with the ON power state and other on the OFF power state.
- The problem is dealt as a regression problem.
- The neural network takes all the points of the graph as the input and outputs the 4 values mentioned above.
- These values are then compared with their Spec values to obtain Compliant/non - Compliant.
- But here the graph contains 10,000 points(features). Also, we just have around 400 samples in ON and OFF dataset each.
- In order to overcome this and avoid overfitting [for the Neural Network to function without overfitting no of features << no of samples] we have reduced the feature space to 30 features using factor analysis.

Data Pre-processing

- After loading the data from points.csv we separate both ON and OFF datasets.

```
ON_list =[]
OFF_list = []
for i in range(len(arr_p)):
    s = arr_p[i][1]
    s = str(s)


    if s.find("N") == -1:
        # APPENDING TO THE OFF LIST
        OFF_list.append(arr_p[i])
```

```
    else:
        ON_list.append(arr_p[i])
```

- We then merge the newly separated data set and the values.csv to form a data structure with both inputs and outputs.

```
df2 = data_v  # data of the values, df1 - data from points,csv
combine = (pd.merge(df1, df2, how='left', on='id'))
```

- We then divide the problem into subproblems one to find the current_stabilised_value (mA), current_max/min_value (mA) and other to find the power_state, current_rise/fall_time_spec (mS).
- To find the solution of the first part of the problem we normalise the data row wise using MinMax Scalar and then divide it into X-train and y-train.

```
X_train = np.concatenate((X_train, y_train), axis=1)
X_train_t = X_train.transpose()

scaler_min_x = MinMaxScaler().fit(X_train_t)


X_min_train = scaler_min_x.transform(X_train_t)

X_min_train = X_min_train.transpose()

Y_min_train = X_min_train[:,10000:10002]
X_min_train = X_min_train[:,0:10000]
```

- The feature space is now reduced to 30 using Factor analysis.

```
from sklearn.decomposition import FactorAnalysis

transformer = FactorAnalysis(n_components=30, random_state=0)
factor_fit = transformer.fit(X_min_train)
X_new = factor_fit.transform(X_min_train)
X_new.shape
```

- The base model of the perceptron :

```
def baseline_model_30(optimizer='adam'):
    # create model
    model = Sequential()
    model.add(Dense(28, activation='relu',
                    kernel_initializer = 'he_normal',
                    input_shape=(30,)))
    model.add(BatchNormalization())
    model.add(Dense(12, activation='relu',
                    kernel_initializer = 'he_normal'))
    model.add(BatchNormalization())

    model.add(Dense(9, activation='relu',
                    kernel_initializer = 'he_normal'))

    model.add(Dense(2, activation='linear',
                    kernel_initializer='he_normal'))
    model.compile(loss = 'mse', optimizer=optimizer, metrics=['mae'])

    return model
```

- We run the OFF/ON model to find the current_stabilised_value (mA) and current_max/min_value (mA) for 200 epochs.

```
model = baseline_model_30()
print (model.get_weights())
print(X_new.shape)
print(y_train.shape)
history = model.fit(X_new,  Y_rob_train, epochs=200, batch_size=5,
verbose=1, validation_split=0.0)
```

- The loss function used is Mean Squared Error and the Optimiser is Adma's Optimiser.
- For the second part of the solution we make use of the outputs produced from solution-1 as the inputs for the solution-2 (to find the values of power_state, current_rise/fall_time_spec (mS).

```
X1_new = np.concatenate((X1,Y1[:,2:4]),axis=1)
print(X1_new.shape)
Y1_new = Y1[:,0:2]
X_train_c= X1_new
y_train_c = Y1_new
```

- We now use RobustScaler for normalising the data column wise.

```
X_rob_train_c = scaler_rob_x.transform(X_train_c)
Y_rob_train_c = scaler_rob_y.transform(y_train_c)
```

- Similar to the solution-1 we use Factor analysis to reduce the feature space.

```
transformer = FactorAnalysis(n_components=30, random_state=0)
factor_fit = transformer.fit(X_rob_train_c[:,0:10000])
X_new1 = factor_fit.transform(X_rob_train_c[:,0:10000])
print(X_new1.shape)
X_new1 = np.concatenate((X_new1,X_rob_train_c[:,10000:10002]),axis=1)
print(X_new1.shape)
```

- The model used for the second half of the solution is :

```
def baseline_model_31(optimizer='adam'):
    # create model
    model = Sequential()
    model.add(Dense(30, activation='relu',
                    kernel_initializer = 'he_normal',
                    input_shape=(32,)))
    model.add(BatchNormalization())
    model.add(Dropout(0.5))

    model.add(Dense(12, activation='relu',
                    kernel_initializer = 'he_normal'))
    model.add(BatchNormalization())
#     model.add(Dropout(0.5))
    model.add(Dense(9, activation='relu',
                    kernel_initializer = 'he_normal'))
    model.add(BatchNormalization())
    model.add(Dense(2, activation='linear',
                    kernel_initializer='he_normal'))
    model.compile(loss = 'mse', optimizer=optimizer, metrics=['mae'])
#     model.summary()
    return model
```

- Unlike the solution-1 we run the solution-2 for 400 epochs with batch size = 5.

```
model1 = baseline_model_31()
history = model1.fit(X_new1,  Y_rob_train_c, epochs=400, batch_size=5,
verbose=1, validation_split=0.0)
```

- Finally all the model weights, details are saved into .h5 files.

```
from keras.backend import manual_variable_initialization
manual_variable_initialization(True)
model.save ("./app/MODEL/my_model_OFF.h5")  # for OFF
model1.save ("./app/MODEL/my_model_1_OFF.h5")   # for OFF
model.save ("./app/MODEL/my_model_ON.h5")  # for ON
model1.save ("./app/MODEL/my_model_1_ON.h5")   # for ON
```

Deploy the model

- To run the code on jupyter notebook install Jupyter Notebook and train the model from scratch create a new python3 console and use the above code by changing the paths for the points.csv and values.csv and the path to store the weights of the model.
- To use the pre-trained model to evaluate the new data use the app [a GUI for the model] in the main repository.

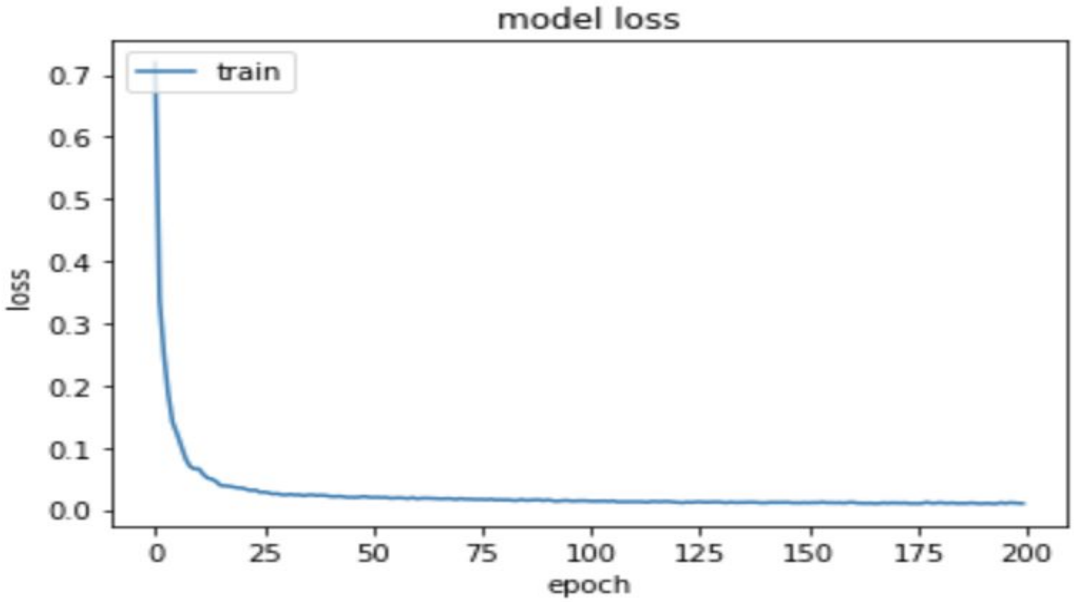
Results

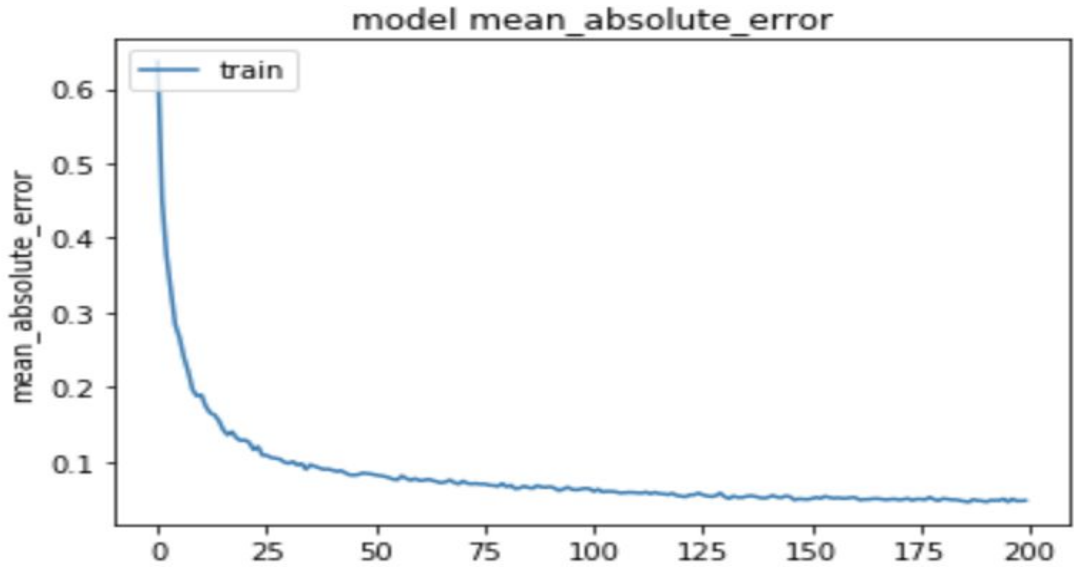
- As we are solving the problem using Regression analysis we use R2 score to measure the efficiency of the model.

R2_SCORES	ON	OFF
current_rise/fall_time_spec (mS).	0.36-0.51	0.45-0.58
current_stabilised_value (mA)	0.78-0.89	0.7-0.8

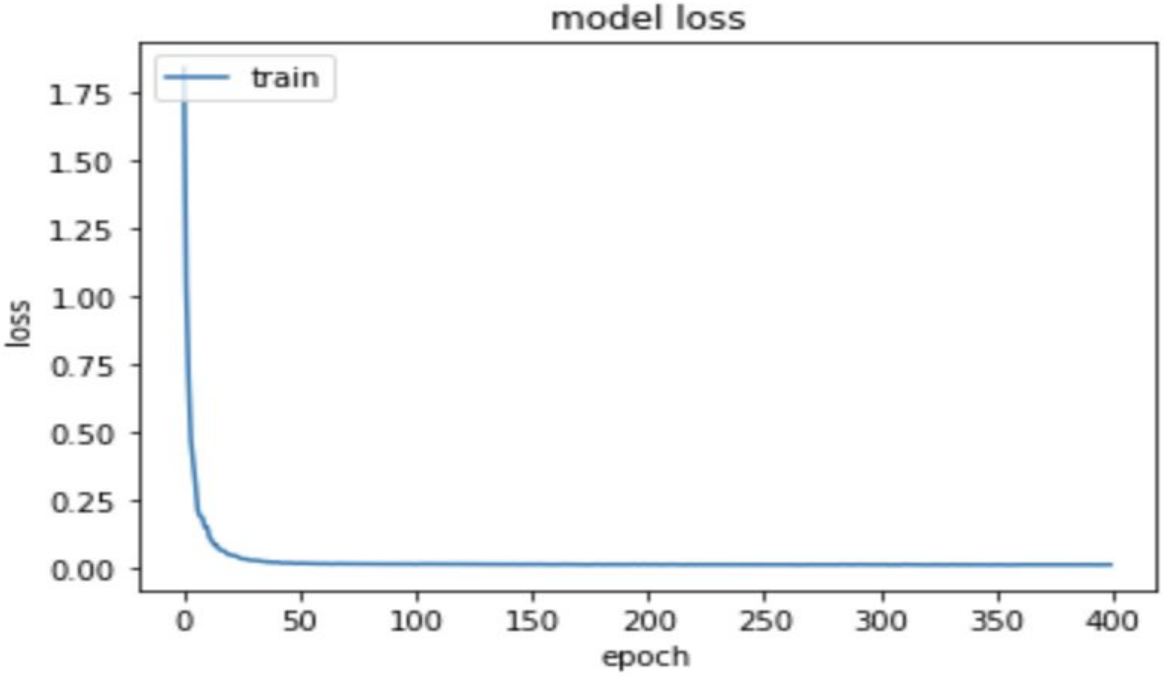
current_max/min_value (mA)	0.57- 0.72	0.6-0.7

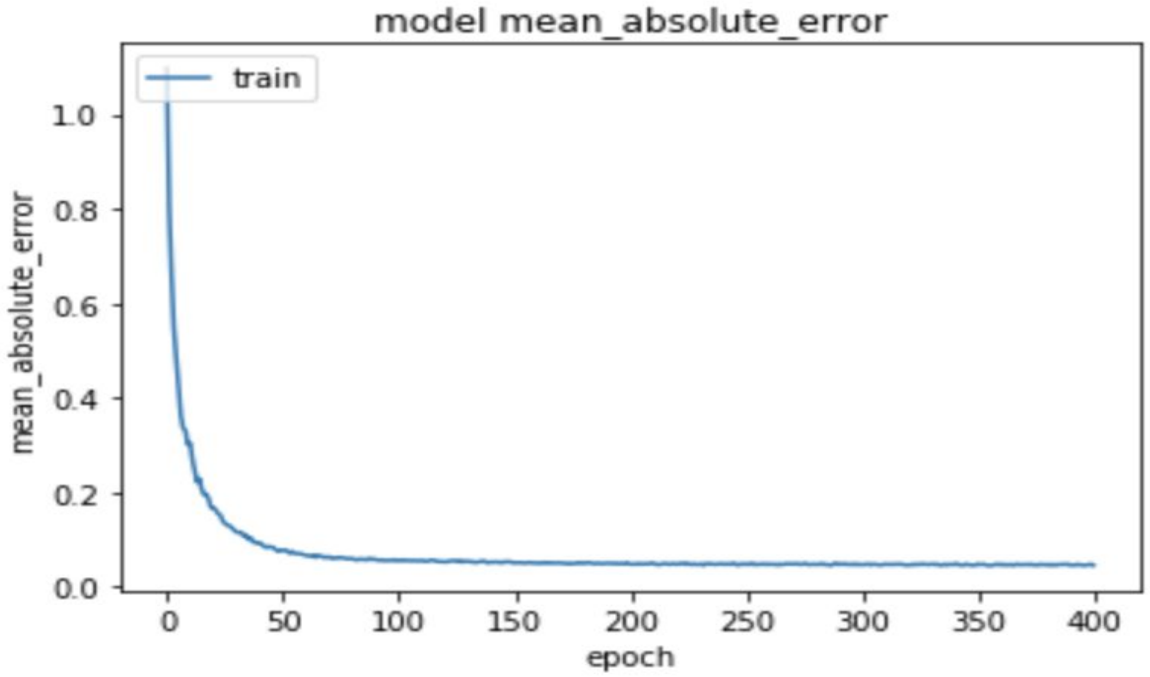
- ON model-1 :



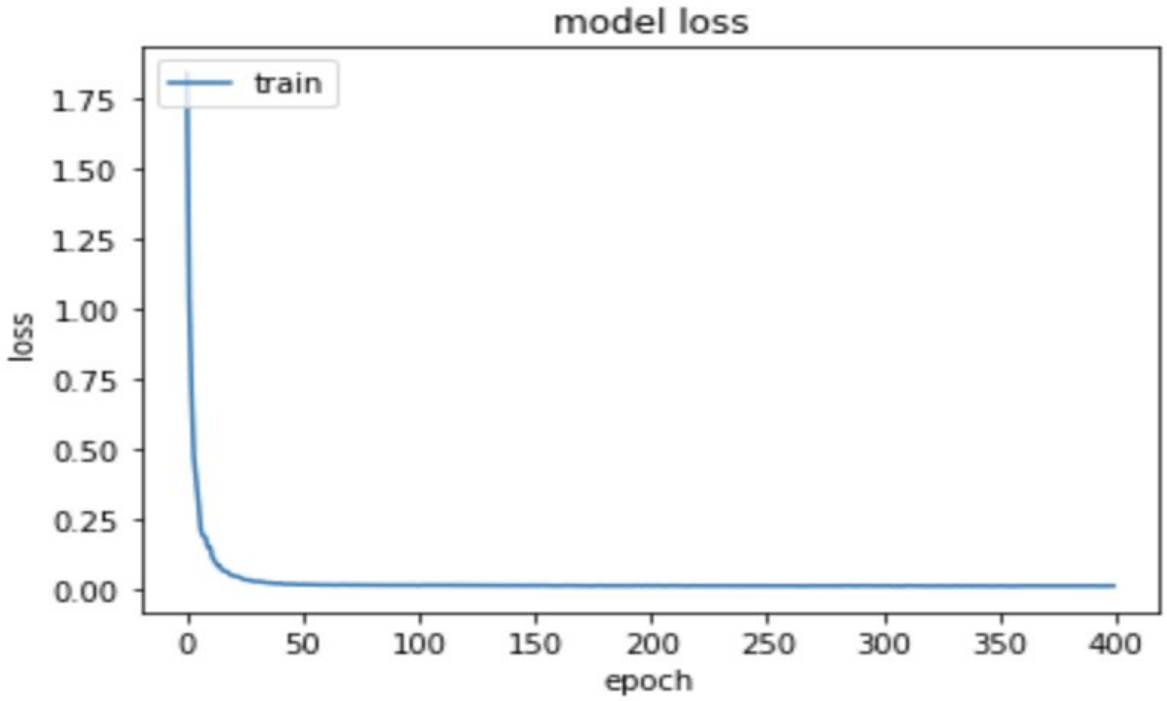


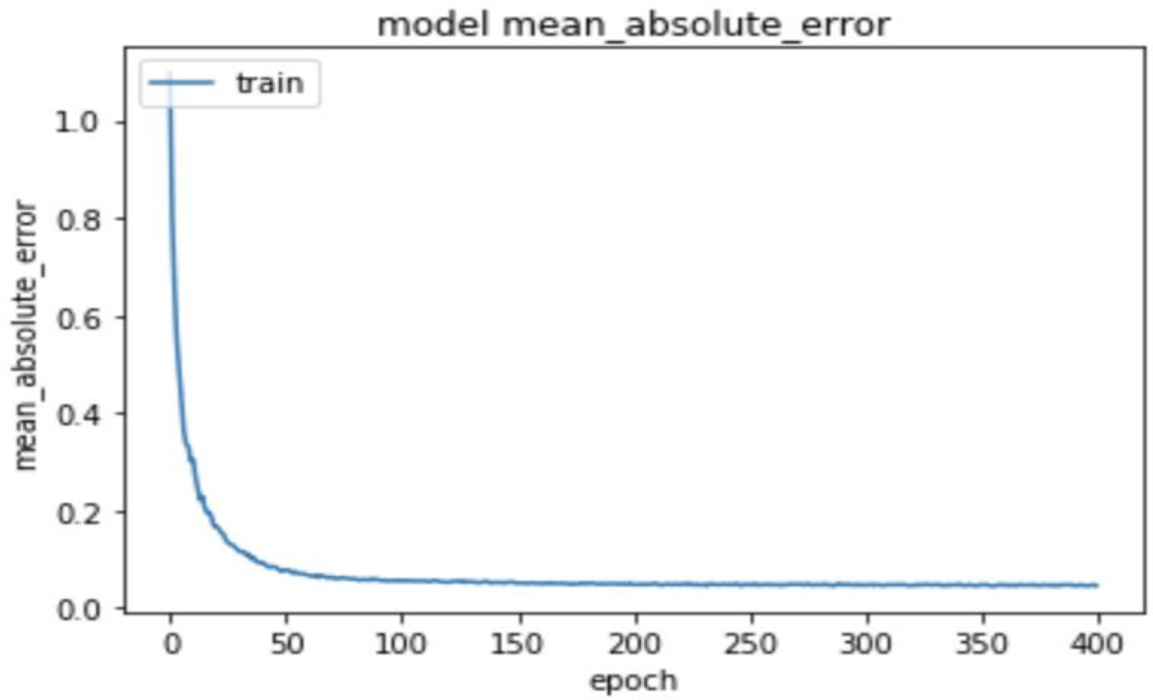
- ON model-2 graphs :



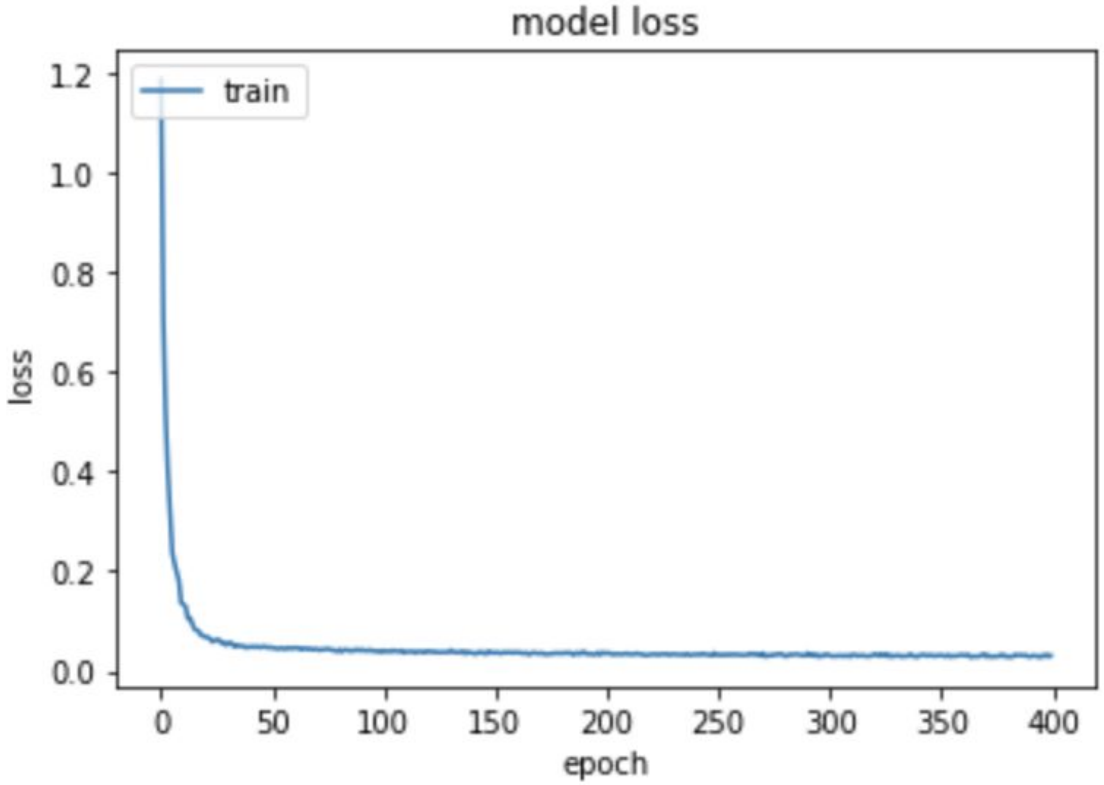


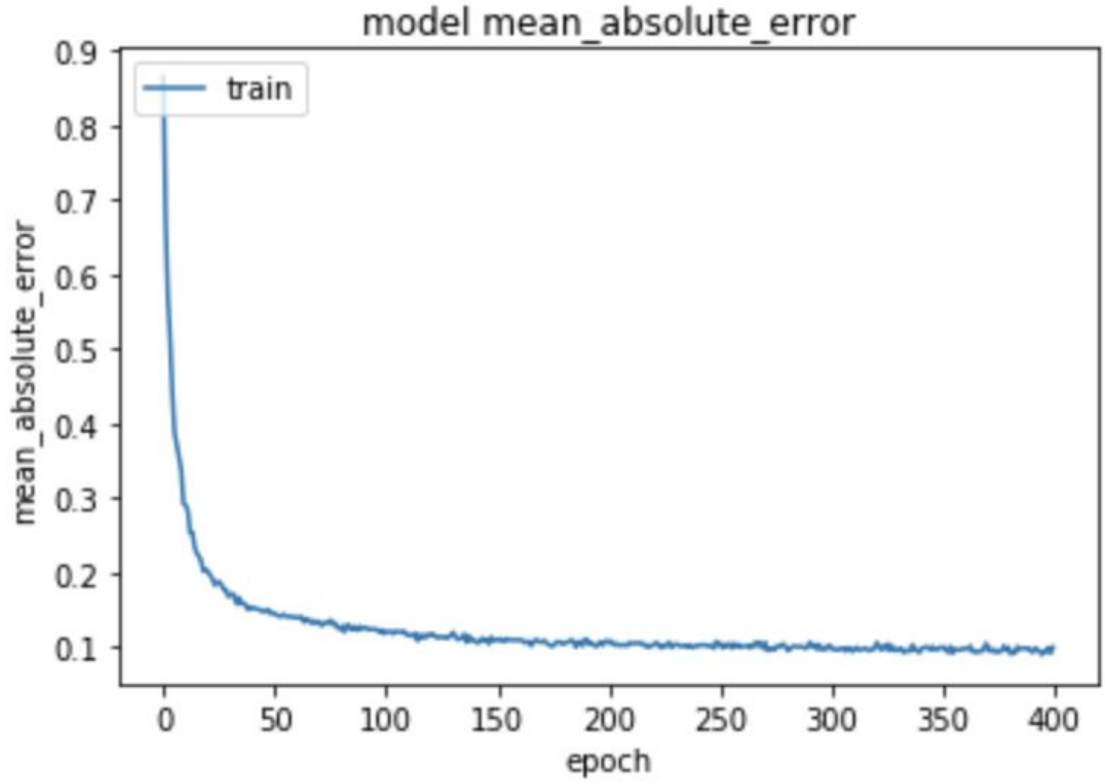
- OFF model - 1 graphs :





- OFF model -2 graphs:





- Final loss function values :

Loss Function values (rmse)	ON	OFF
Model - 1	0.01660	0.024242
Model - 2	0.0097815	0.027379

Conclusions

- The low accuracy is due to the availability of a small training data set.
- So, in future the present implementation can be extended when a huge amount of dataset is produced.
- The other main reason for the low accuracy is the reduction of the feature space. This can also be avoided with huge amount of datasets.