The 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 with 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/not - Compliant.
* But here the graph contains 10,000 points( features ). Also, we just have around 400 samples in ON and OFF dataset each.
* 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 and Code Analysis

* 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 normalize 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 normalizing 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

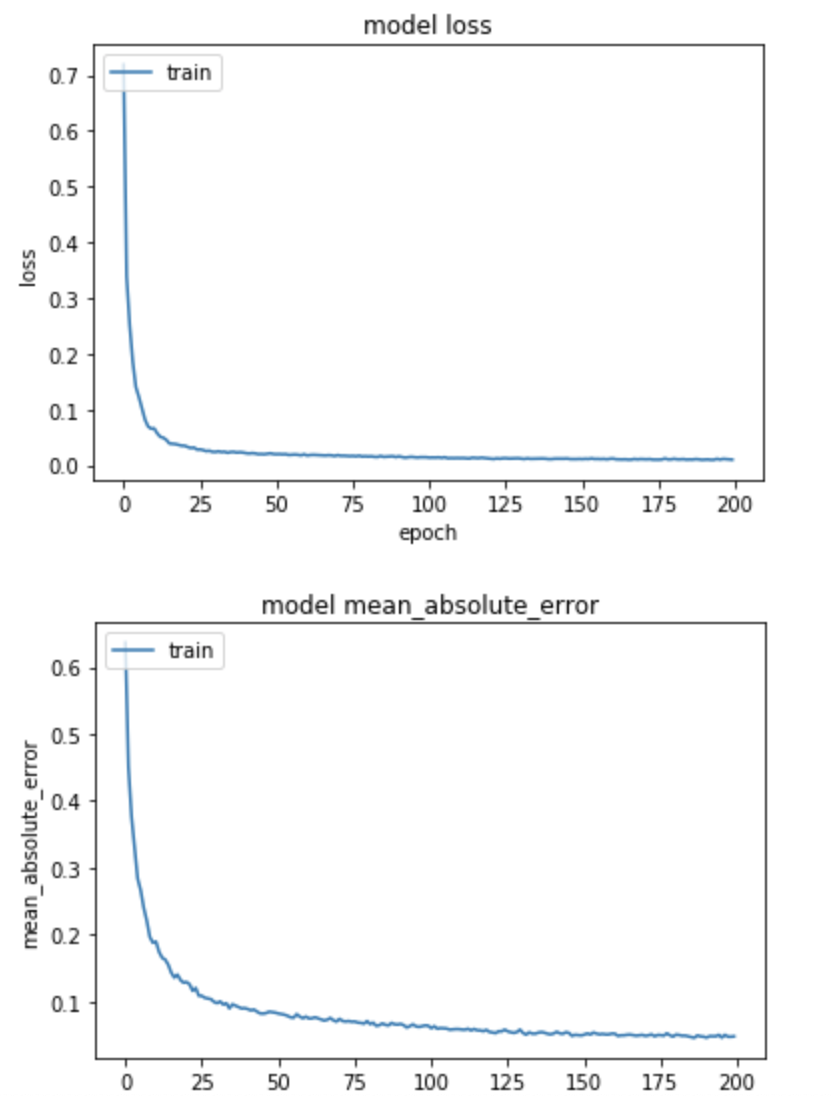
* To run the code on jupyter notebook install Jupyter Notebook and train the model from scratch, create a new python3 console and use the above code by changing the paths for the points.csv and values.csv and the path to store the weights of the model.
* To use the pre-trained model to evaluate the new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

Results

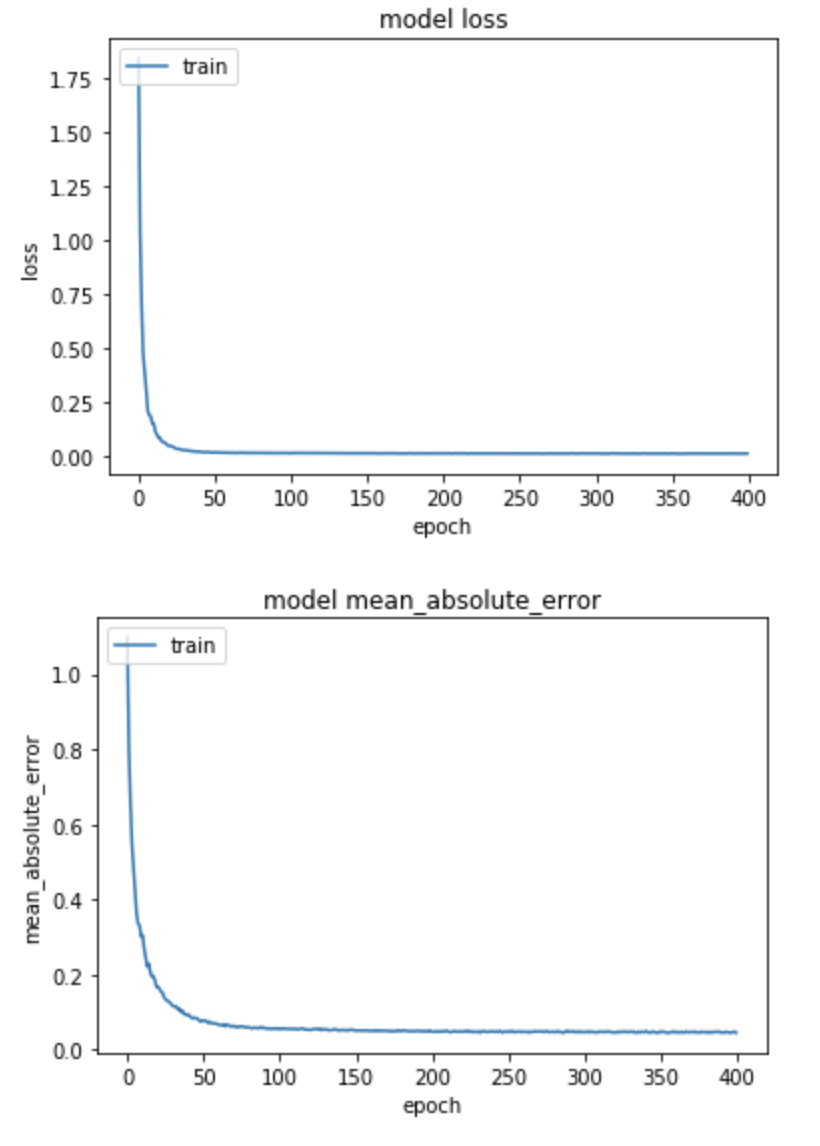
* As we are solving the problem using Regression analysis we use R2 score to measure the efficiency of the model.

|  |  |  |
| --- | --- | --- |
| 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 |

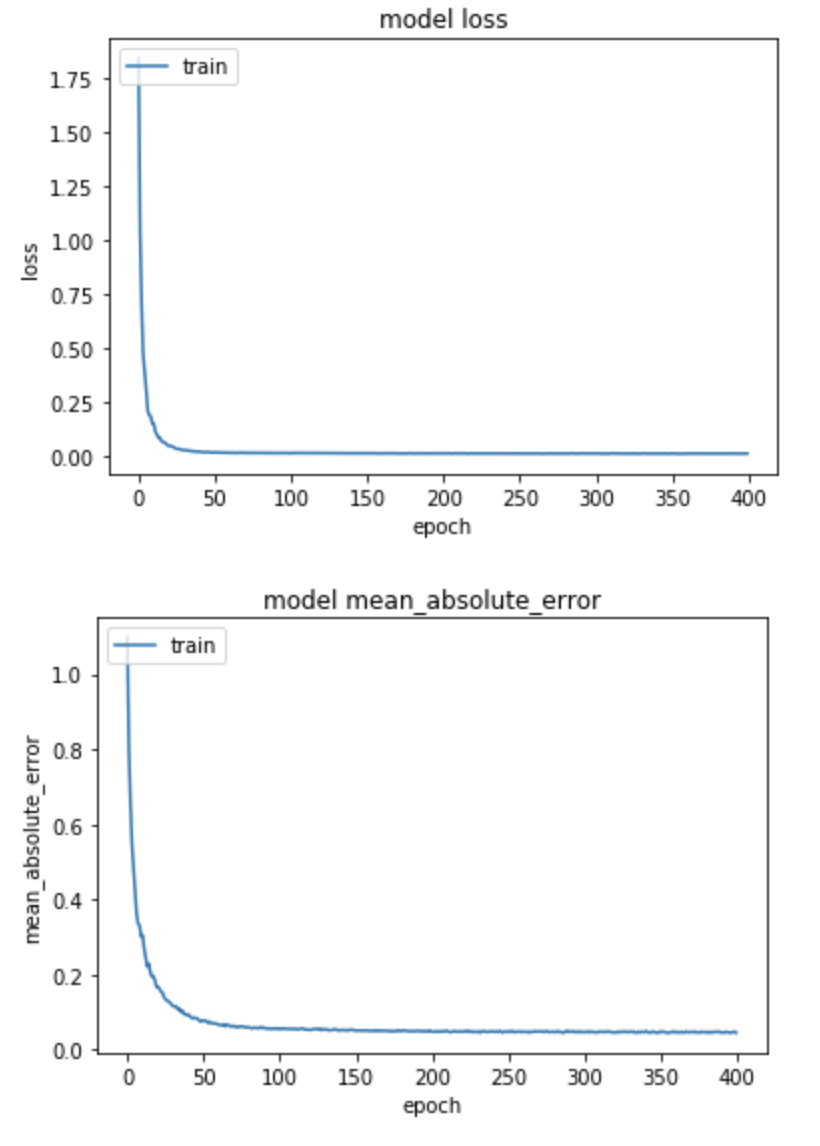
* ON model-1 :

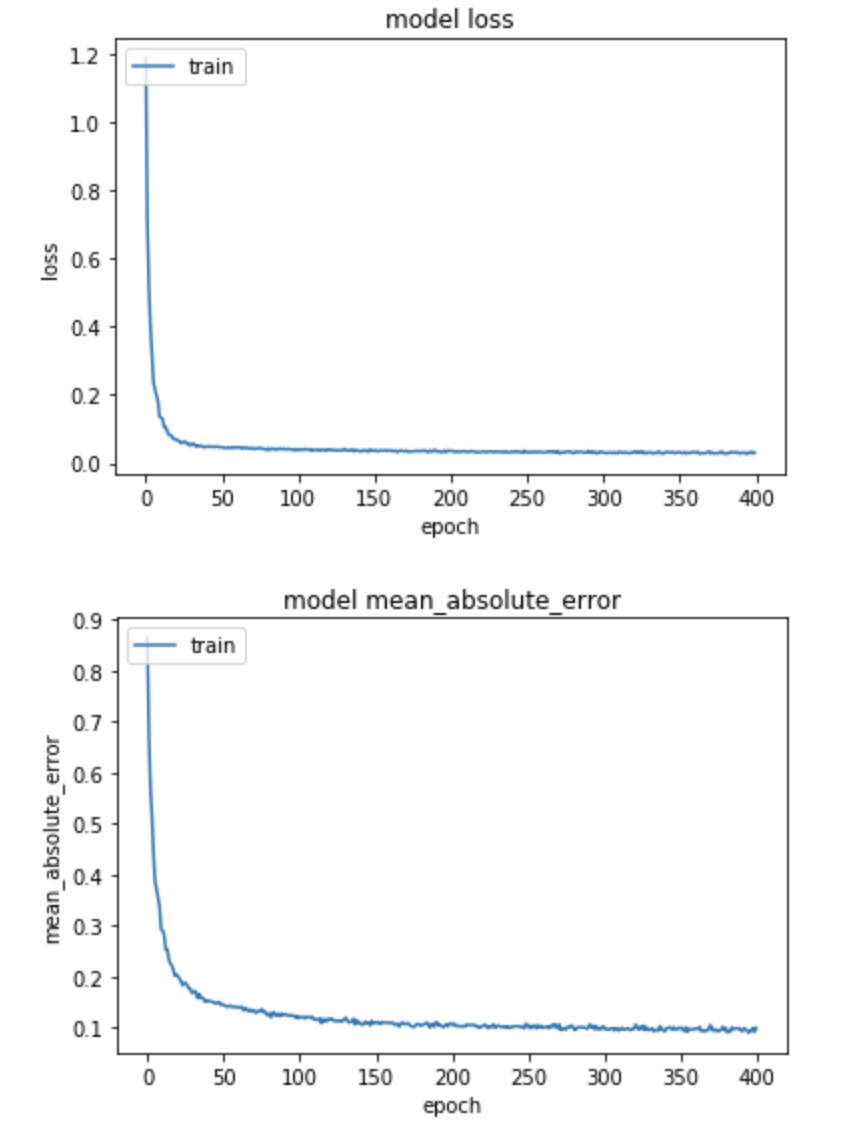


* ON model-2 graphs :



* 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 |

* R2 scores - It is the proportion of the variance in the dependent variable that is predictable from the independent variable(s). It is (Total variance explained by the model) / total variance. A low value would show a low level of correlation, mean
* R-squared does not indicate whether a regression model is adequate. You can have a low R-squared value for a good model, or a high R-squared value for a model that does not fit the data.
* R-squared provides an estimate of the strength of the relationship between your model and the response variable

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 a huge amount of datasets.
* We can also see that the R2 score is highly affected by outliers in the data.

References

[1] C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.

[2]Lifeng He, Xiwei Ren, Qihang Gao, Xiao Zhao, Bin Yao, Yuyan Chao [The connected-component labeling problem: A review of state-of-the-art algorithms](https://www.sciencedirect.com/science/article/pii/S0031320317301693)

[3]T. Warren Liao [Clustering of time series data—a survey](https://www.sciencedirect.com/science/article/pii/S0031320305001305)

[4] Dunja Mladenić Feature Selection for Dimensionality Reduction

[5]<https://www.analyticsvidhya.com/blog/2018/08/dimensionality-reduction-techniques-python/>

[6] <https://en.wikipedia.org/wiki/Principal_component_analysis>

[7<https://github.com/GKarmakar/RegressionUsingNN/blob/master/RegressionUsingNeuralNetwork.ipynb>

[8]<https://github.com/buomsoo-kim/Easy-deep-learning-with-Keras>

[9]<https://www.tensorflow.org/guide/keras>

[10][https://www.kaggle.com](https://www.kaggle.com/)