

# Deep Learning and Reinforcement Learning

IBM Machine Learning - Project  
Mohammed Qasim K.  
November 2022



## Main Objective

- The main objective of this analysis was to predict the torque and temperatures of various elements in a permanent magnet synchronous motor (PMSM) using a Ensemble Neural Net (ENN) and Recurrent Neural Net (RNN).
- The data set was split into Training set (40%), Validation set (60%) for the ENN.
- The data set was split into Training set (40%), Validation set (10%) for the RNN.
- Data with *profile\_id = 20* was used as the Test set for All Cases



## About the Data

- The data set comprises of several sensor data collected from a permanent magnet synchronous motor (PMSM) deployed on a test bench. Test bench measurements were collected by the LEA department at Paderborn University.
- This data set has 1330816 records and 13 variables. During the analysis, no duplicates were detected.

<https://www.kaggle.com/datasets/wkirgsn/electric-motor-temperature>

Variable name	Type	Description
u_q	float	Voltage q-component measurement in dq-coordinates (in V)
coolant	float	Coolant temperature (in °C)
stator_winding	float	Stator winding temperature (in °C) measured with thermocouples
u_d	float	Voltage d-component measurement in dq-coordinates
stator_tooth	float	Stator tooth temperature (in °C) measured with thermocouples
motor_speed	float	Motor speed (in rpm)
i_d	float	Current d-component measurement in dq-coordinates
stator_yoke	float	Stator yoke temperature (in °C) measured with thermocouples
i_q	float	Current q-component measurement in dq-coordinates
ambient	float	Ambient temperature (in °C)
torque	float	Motor torque (in Nm)
profile_id	int	Measurement session id. Each distinct measurement session can be identified through this integer id.
pm	float	Permanent magnet temperature (in °C) measured with thermocouples and transmitted wirelessly



# Data Exploration

- The target variable were *stator\_winding*, *stator\_tooth*, *stator\_yoke* and *torque*.
- After checking for duplicates, the EDA was conducted on the Data set.
- The data description is as follows:

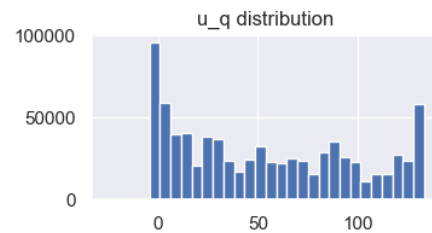
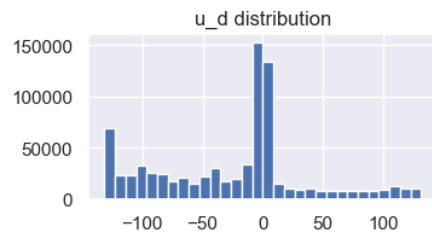
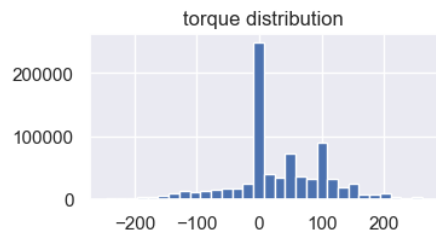
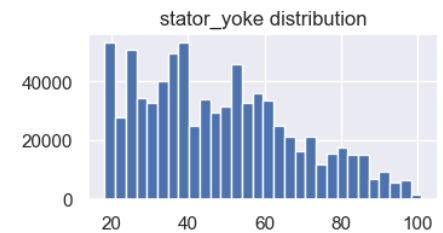
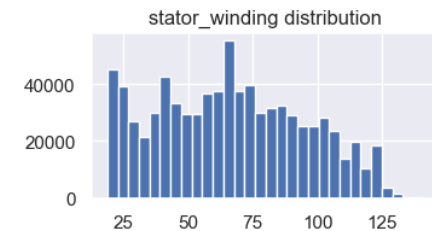
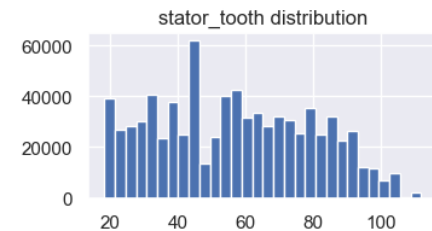
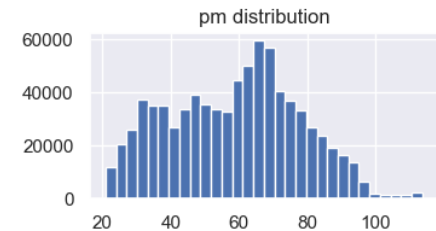
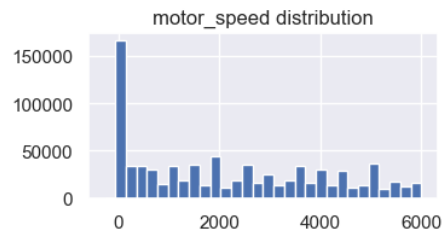
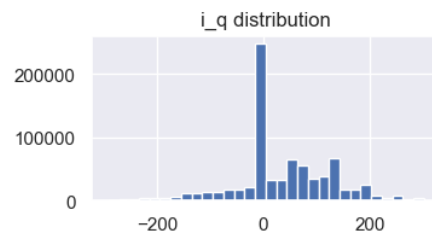
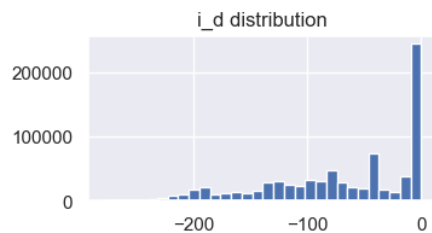
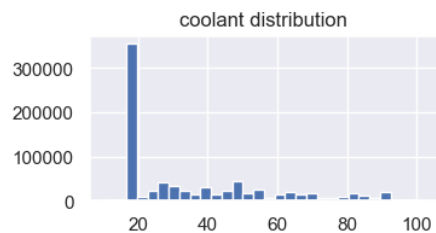
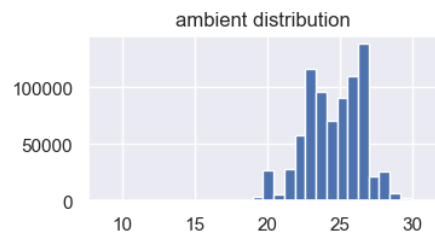
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1330816 entries, 0 to 1330815
Data columns (total 13 columns):
#   Column          Non-Null Count  Dtype
---  -
0   u_q              1330816 non-null float64
1   coolant          1330816 non-null float64
2   stator_winding   1330816 non-null float64
3   u_d              1330816 non-null float64
4   stator_tooth     1330816 non-null float64
5   motor_speed      1330816 non-null float64
6   i_d              1330816 non-null float64
7   i_q              1330816 non-null float64
8   pm               1330816 non-null float64
9   stator_yoke      1330816 non-null float64
10  ambient          1330816 non-null float64
11  torque           1330816 non-null float64
12  profile_id       1330816 non-null int64
dtypes: float64(12), int64(1)
memory usage: 132.0 MB
```



# Data Exploration

- Dropped *profile\_id* column.

	u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_	ambient	torque
count	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06	1.330816e+06
mean	5.427900e+01	3.622999e+01	6.634275e+01	-2.513381e+01	5.687858e+01	2.202081e+03	-6.871681e+01	3.741278e+01	5.850678e+01	4.818791e+01	2.456526e+01	3.110603e+01
std	4.417323e+01	2.178615e+01	2.867206e+01	6.309197e+01	2.295223e+01	1.859663e+03	6.493323e+01	9.218188e+01	1.900150e+01	1.999101e+01	1.929522e+00	7.713575e+01
min	-2.529093e+01	1.062375e+01	1.858582e+01	-1.315304e+02	1.813398e+01	-2.755491e+02	-2.780036e+02	-2.934268e+02	2.085696e+01	1.807661e+01	8.783478e+00	-2.464667e+02
25%	1.206992e+01	1.869814e+01	4.278796e+01	-7.869090e+01	3.841601e+01	3.171107e+02	-1.154061e+02	1.095863e+00	4.315158e+01	3.199031e+01	2.318480e+01	-1.374265e-01
50%	4.893818e+01	2.690014e+01	6.511013e+01	-7.429755e+00	5.603635e+01	1.999977e+03	-5.109376e+01	1.577401e+01	6.026629e+01	4.562551e+01	2.479733e+01	1.086035e+01
75%	9.003439e+01	4.985749e+01	8.814114e+01	1.470271e+00	7.558668e+01	3.760639e+03	-2.979688e+00	1.006121e+02	7.200837e+01	6.146081e+01	2.621702e+01	9.159718e+01
max	1.330370e+02	1.015985e+02	1.413629e+02	1.314698e+02	1.119464e+02	6.000015e+03	5.189670e-02	3.017079e+02	1.136066e+02	1.011481e+02	3.071420e+01	2.610057e+02





# Model Variations

- Four different Neural Net Models were created with varying activation functions (Sigmoid, ReLU & Linear) and Dense Layers (16 or 32).
- The optimizer used was adam the loss metrics used were either Mean Squared Error (MSE) or Mean Absolute Percentage Error (MAPE) .
- The output from the Neural Net Models were of dimension (Number of Training Values, 4).

```
input_1 = Input(shape=(X.shape[1]))
hlay1_1 = Dense(16,activation = 'relu')(input_1)
hlay2_1 = Dense(16,activation = 'relu')(hlay1_1)
final_dnn1 = Dense(4,activation = 'linear')(hlay2_1)
dnn1 = Model(inputs=input_1, outputs=final_dnn1)
dnn1.compile(optimizer = "adam", loss = "mean_squared_error")

input_2 = Input(shape=(X.shape[1]))
hlay1_2 = Dense(32,activation = 'relu')(input_2)
hlay2_2 = Dense(16,activation = 'relu')(hlay1_2)
final_dnn2 = Dense(4,activation = 'linear')(hlay2_2)
dnn2 = Model(inputs=input_2, outputs=final_dnn2)
dnn2.compile(optimizer = "adam", loss = "mean_absolute_percentage_error")

input_3 = Input(shape=(X.shape[1]))
hlay1_3 = Dense(16,activation = 'sigmoid')(input_3)
hlay2_3 = Dense(16,activation = 'sigmoid')(hlay1_3)
hlay3_3 = Dense(8,activation = 'relu')(hlay2_3)
final_dnn3 = Dense(4,activation = 'linear')(hlay3_3)
dnn3 = Model(inputs=input_3, outputs=final_dnn3)
dnn3.compile(optimizer = "adam", loss = "mean_squared_error")

input_4 = Input(shape=(X.shape[1]))
hlay1_4 = Dense(32,activation = 'relu')(input_4)
hlay2_4 = Dense(32,activation = 'sigmoid')(hlay1_4)
final_dnn4 = Dense(4,activation = 'linear')(hlay2_4)
dnn4 = Model(inputs=input_4, outputs=final_dnn4)
dnn4.compile(optimizer = "adam", loss = "mean_squared_error")
```



# Model Variations

- The Four different Neural Net Models were used as non trainable inputs into one ENN.
- The ENN was created with varying activation functions (ReLU & Linear) and Dense Layers (8 or 32).
- The optimizer used was *adam* the loss metrics used was Mean Squared Error (MSE).
- The output from the Neural Net Models was of dimension (Number of Training Values, 4).

```
dnn1.trainable = False; dnn2.trainable = False; dnn3.trainable = False; dnn4.trainable = False;

X_train, X_val, y_train, y_val = train_test_split(X,y, test_size = 0.2)

enn_input = Input(shape=(X_train.shape[1]))
const_1 = dnn1(enn_input)
const_2 = dnn2(enn_input)
const_3 = dnn3(enn_input)
const_4 = dnn4(enn_input)
enn_1 = Dense(8,activation = 'relu')(enn_input)
merge = concatenate(inputs = [const_1,const_2,const_3,const_4,enn_1])
enn_2 = Dense(32,activation = 'relu')(merge)
enn_3 = Dense(32,activation = 'relu')(enn_2)
final = Dense(4,activation = 'linear')(enn_3)
enn = Model(inputs=enn_input, outputs=final)
enn.compile(optimizer = "adam", loss = "mean_squared_error")
```






## Model Variations

- The ENN was used as a non trainable input into one RNN.
- The RNN was created with ReLU activation functions and 16 Dense Layers.
- Simple RNN & Dropout Layers was also used.
- The optimizer used was *adam* the loss metrics used was Mean Squared Error (MSE).
- The output from the RNN was of dimension (Number of Training Values, 4).

```
enn.trainable = False;

input_rnn = Input(shape=(X_train.shape[1], 1))
input_enn = Input(shape=(X_train.shape[1]))
rnn1 = SimpleRNN(16, return_sequences=True)(input_rnn)
drop1 = Dropout(0.2)(rnn1)
rnn2 = SimpleRNN(16, return_sequences=False)(drop1)
drop2 = Dropout(0.2)(rnn2)
enn_layer = enn(input_enn)
merge = concatenate(inputs = [enn_layer, drop2])
combine1 = Dense(16, activation = 'relu')(merge)
combine2 = Dense(16, activation = 'relu')(combine1)
final = Dense(4)(combine2)

ernn = Model(inputs=[input_rnn, input_enn], outputs=final)
ernn.compile(optimizer = "adam", loss = "mean_squared_error")
```



## Cross-Validation and Regularization

- Iterations over different polynomial degrees (1, 2,) and alphas.
- Results are sorted by RMSE in ascending order.

### Linear

	Average RMSE
Degree = 3	1590.404635
Degree = 2	1612.824436
Degree = 1	1716.996101
Degree = 4	2080.145656
Degree = 5	8573.970563
Degree = 6	167125.925844

### Ridge

	Average RMSE
Degree = 3, alpha = 0.005	1590.408087
Degree = 3, alpha = 0.01	1590.411623
Degree = 3, alpha = 0.05	1590.442776
Degree = 3, alpha = 0.1	1590.488293
Degree = 3, alpha = 0.3	1590.729348

### Lasso

	Average RMSE
Degree = 3, alpha = 0.3	1589.662468
Degree = 3, alpha = 0.1	1589.918209
Degree = 3, alpha = 0.05	1590.094432
Degree = 3, alpha = 0.01	1590.336092
Degree = 3, alpha = 0.005	1590.369968

### Elastic Net

	Average RMSE
Degree = 3, alpha = 0.005	1659.788222
Degree = 2, alpha = 0.005	1662.664297
Degree = 2, alpha = 0.01	1698.958361
Degree = 3, alpha = 0.01	1703.922403
Degree = 1, alpha = 0.005	1797.426905



## Cross-Validation and Regularization

- The error metrics between ENN and RNN were not significantly different.
- ENN performed better although further training could be done on both models to see the maximum possible scores.

ENN

Training Set R2 Score: 0.9355116520614097

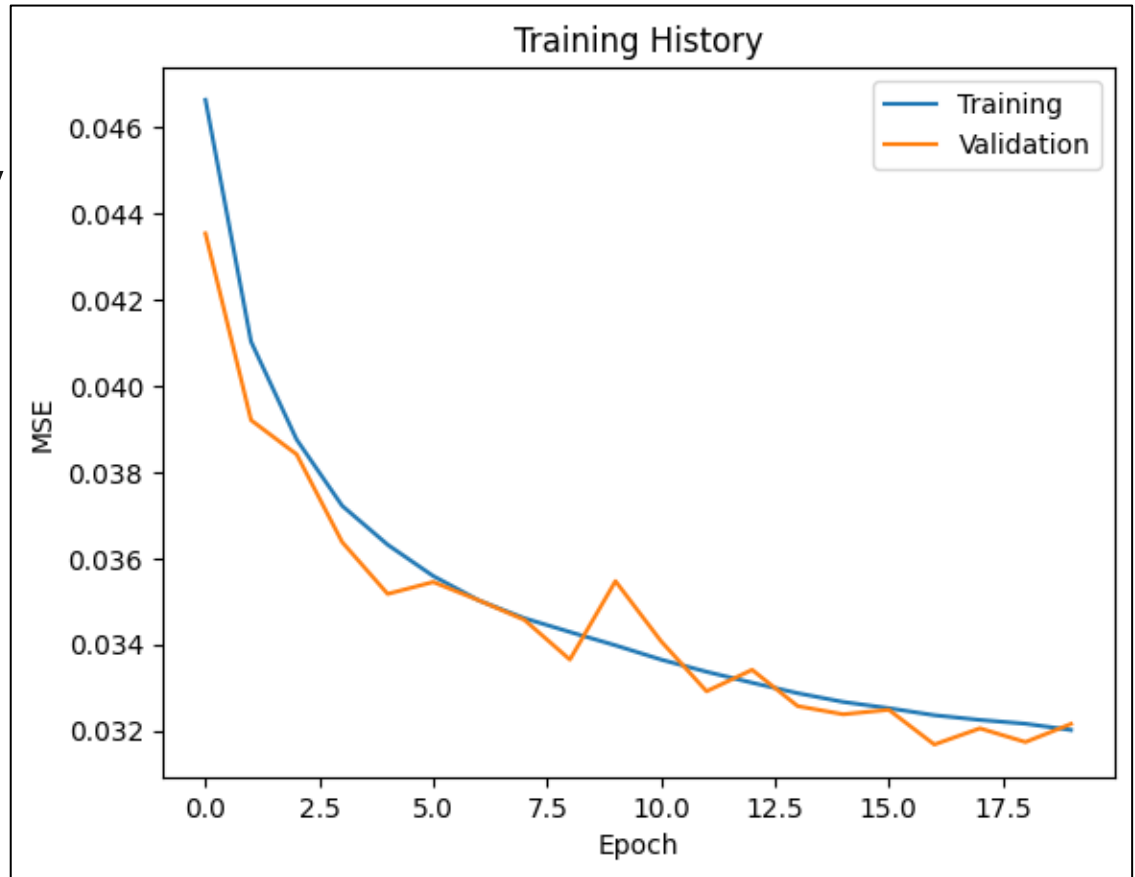
Test Set R2 Score: 0.9254317790958869

RNN

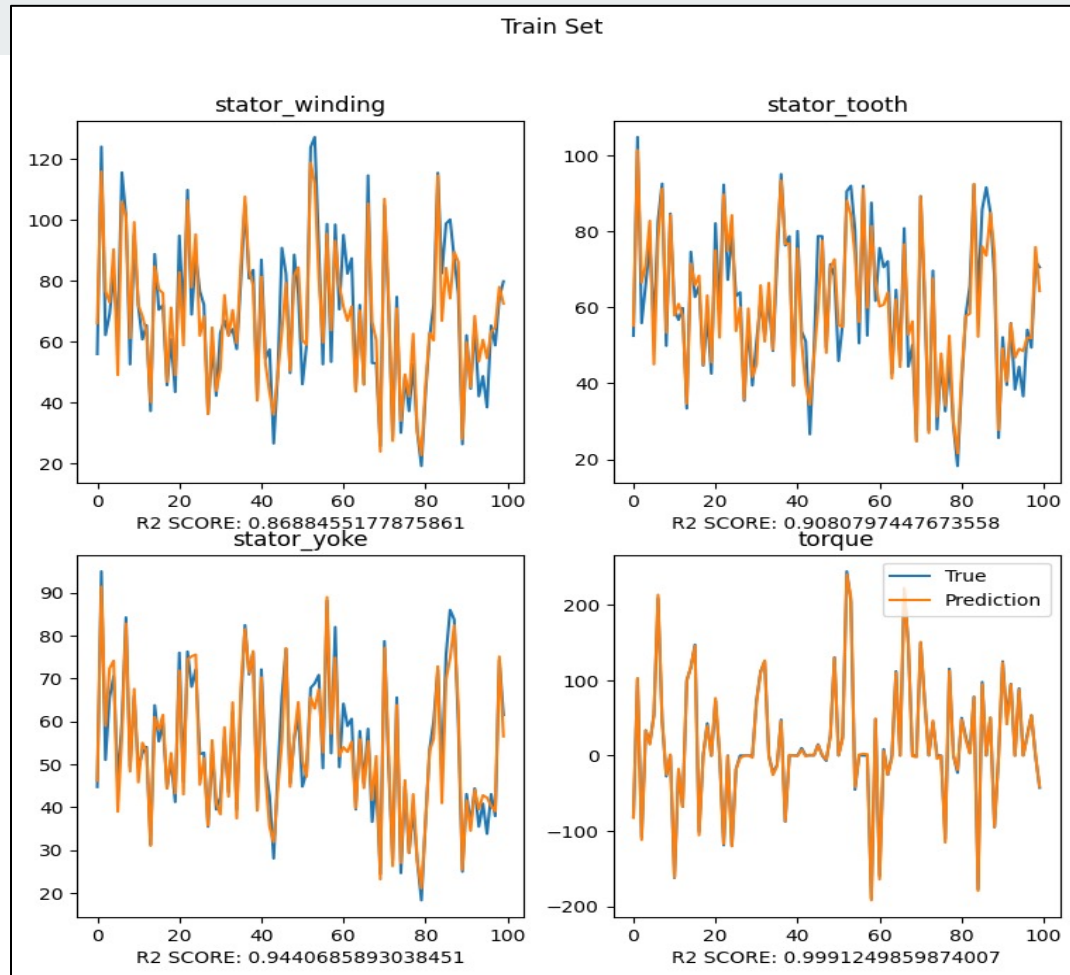
Training Set R2 Score: 0.9223374430056294

Test Set R2 Score: 0.9176434940000786

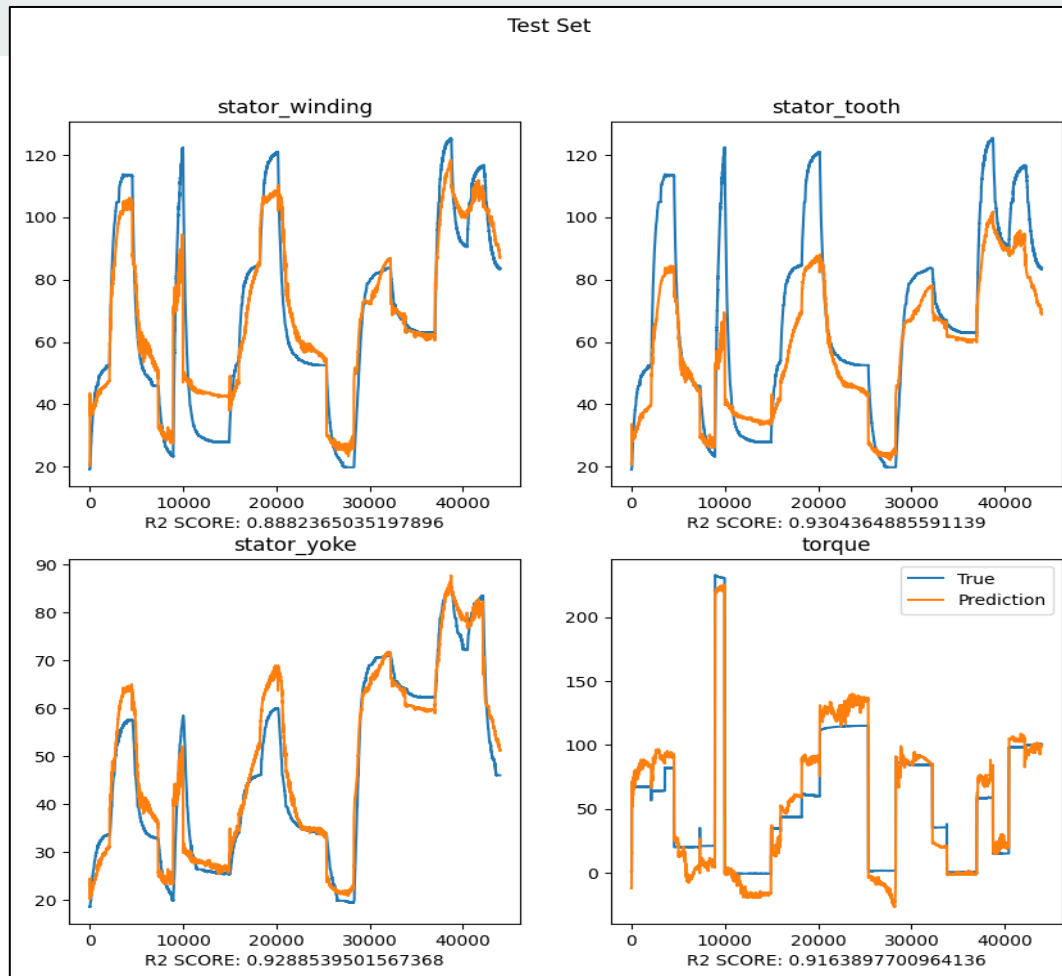
## Training History (ENN)



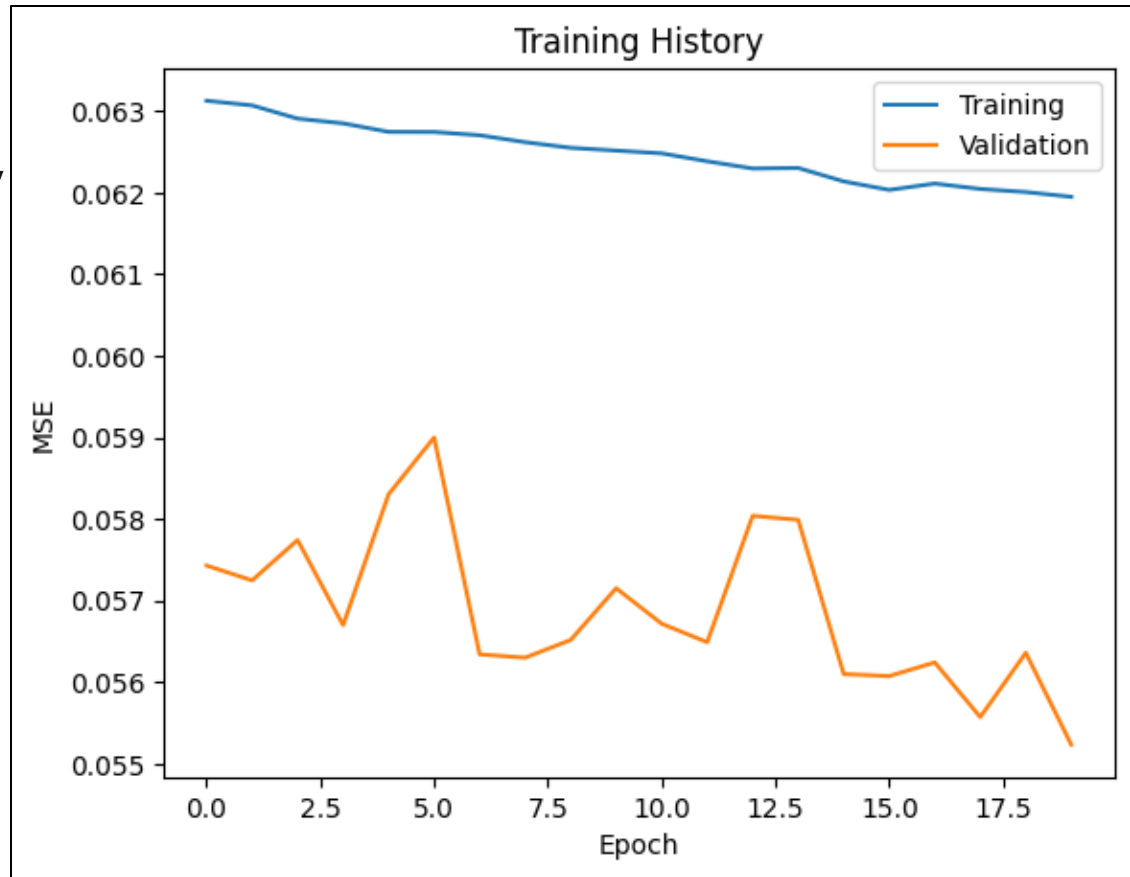
# Predictions on the Training Set (ENN)



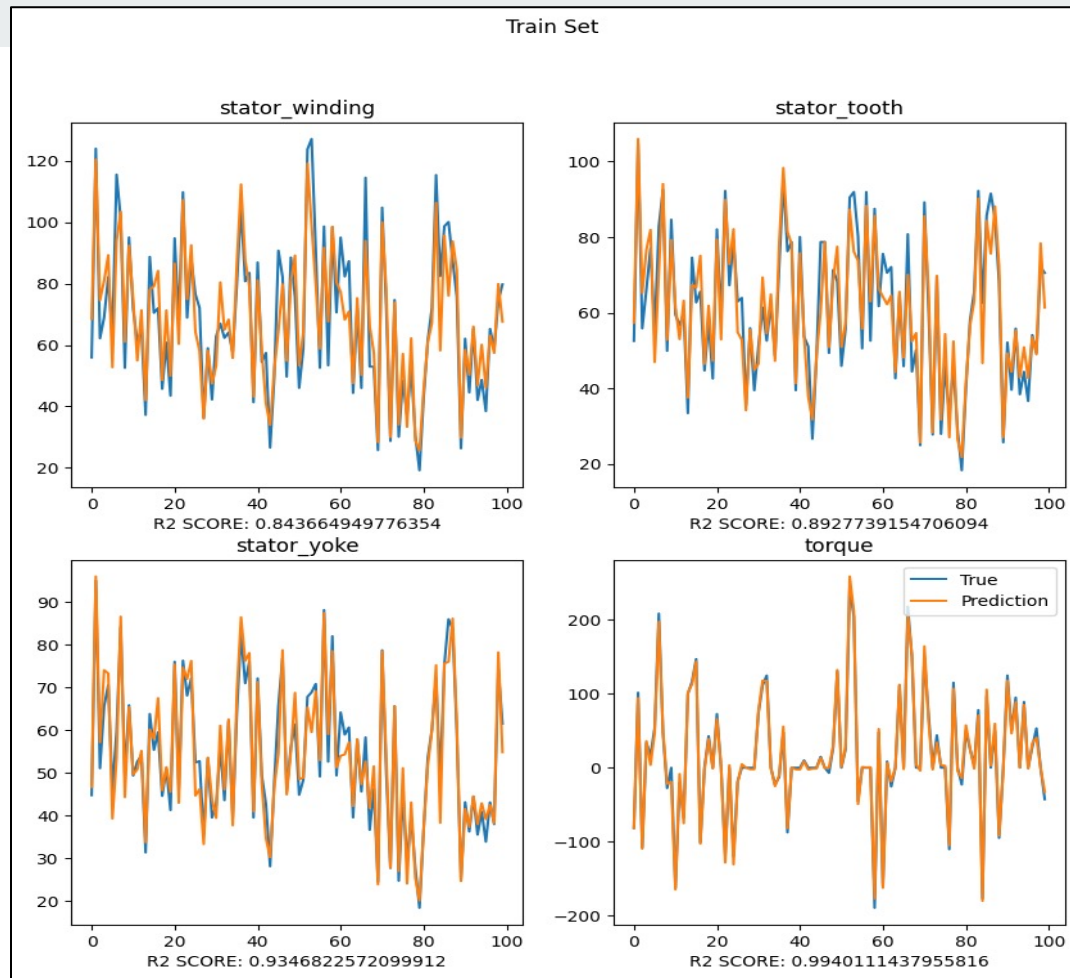
# Predictions on the Test Set (ENN)



## Training History (RNN)

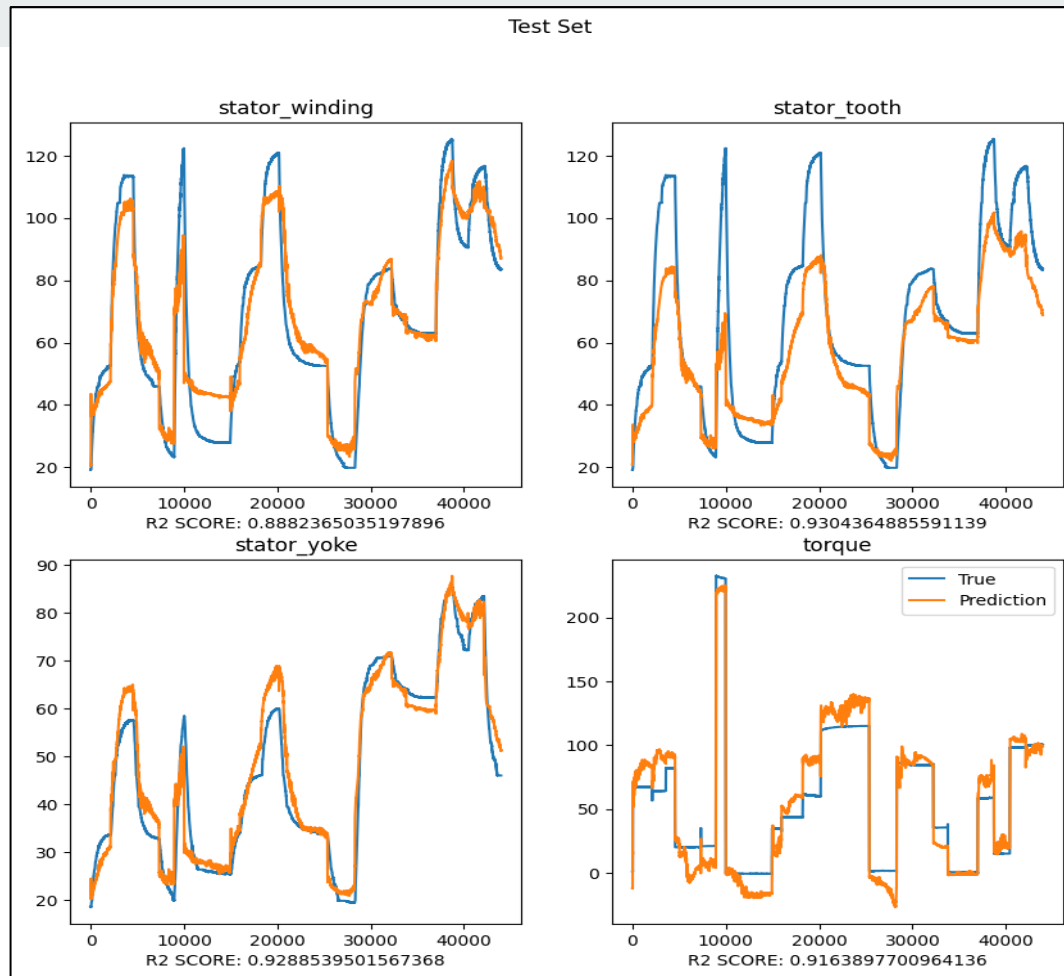


# Predictions on the Training Set (RNN)





# Predictions on the Test Set (RNN)





## Conclusion

The analysis shows that Model Variation has a large effect on the model performance. In this case, ENN performed although further training could be done on both models to see the maximum possible scores.. The ENN had the highest R2 value when predicting on the test set.

Jupyter Notebooks & Models can be found here:

<https://github.com/moqa19/IBM-Machine-Learning/blob/main/ML%205.ipynb>

<https://github.com/moqa19/IBM-Machine-Learning/blob/main/ML%205%20Models.ipynb>

<https://github.com/moqa19/IBM-Machine-Learning/tree/main/ML%205%20Models>