Deep Learning for Predictive Maintenance

In [41]:
```
import keras
```

In [42]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.layers import Dense, Dropout, GRU, Input,Conv1D, MaxPooling1D, Flatten, concatenate
%matplotlib inline
from keras.models import Model
```

In [43]:
```
train_data = pd.read_csv('train_data.txt')
train_data
```

Out[43]:

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s16	s17	s18	s19	s20	s21	RUL	label1	label2	cycle_norm
0	1	1	0.459770	0.166667	0.0	0.0	0.183735	0.406802	0.309757	0.0	...	0.0	0.333333	0.0	0.0	0.713178	0.724662	191	0	0	0.000000
1	1	2	0.609195	0.250000	0.0	0.0	0.283133	0.453019	0.352633	0.0	...	0.0	0.333333	0.0	0.0	0.666667	0.731014	190	0	0	0.002770
2	1	3	0.252874	0.750000	0.0	0.0	0.343373	0.369523	0.370527	0.0	...	0.0	0.166667	0.0	0.0	0.627907	0.621375	189	0	0	0.005540
3	1	4	0.540230	0.500000	0.0	0.0	0.343373	0.256159	0.331195	0.0	...	0.0	0.333333	0.0	0.0	0.573643	0.662386	188	0	0	0.008310
4	1	5	0.390805	0.333333	0.0	0.0	0.349398	0.257467	0.404625	0.0	...	0.0	0.416667	0.0	0.0	0.589147	0.704502	187	0	0	0.011080
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
20626	100	196	0.477011	0.250000	0.0	0.0	0.686747	0.587312	0.782917	0.0	...	0.0	0.750000	0.0	0.0	0.271318	0.109500	4	1	2	0.540166
20627	100	197	0.408046	0.083333	0.0	0.0	0.701807	0.729453	0.866475	0.0	...	0.0	0.583333	0.0	0.0	0.124031	0.366197	3	1	2	0.542936
20628	100	198	0.522989	0.500000	0.0	0.0	0.665663	0.684979	0.775321	0.0	...	0.0	0.833333	0.0	0.0	0.232558	0.053991	2	1	2	0.545706
20629	100	199	0.436782	0.750000	0.0	0.0	0.608434	0.746021	0.747468	0.0	...	0.0	0.583333	0.0	0.0	0.116279	0.234466	1	1	2	0.548476
20630	100	200	0.316092	0.083333	0.0	0.0	0.795181	0.639634	0.842167	0.0	...	0.0	0.666667	0.0	0.0	0.178295	0.218172	0	1	2	0.551247

20631 rows × 30 columns

In [44]:
```
test_data = pd.read_csv('test_data.txt')
test_data
```

Out[44]:

	id	cycle	setting1	setting2	setting3	s1	s2	s3	s4	s5	...	s16	s17	s18	s19	s20	s21	cycle_norm	RUL	label1	label2
0	1	1	0.632184	0.750000	0.0	0.0	0.545181	0.310661	0.269413	0.0	...	0.0	0.333333	0.0	0.0	0.558140	0.661834	0.000000	142	0	0
1	1	2	0.344828	0.250000	0.0	0.0	0.150602	0.379551	0.222316	0.0	...	0.0	0.416667	0.0	0.0	0.682171	0.686827	0.002770	141	0	0
2	1	3	0.517241	0.583333	0.0	0.0	0.376506	0.346632	0.322248	0.0	...	0.0	0.416667	0.0	0.0	0.728682	0.721348	0.005540	140	0	0
3	1	4	0.741379	0.500000	0.0	0.0	0.370482	0.285154	0.408001	0.0	...	0.0	0.250000	0.0	0.0	0.666667	0.662110	0.008310	139	0	0
4	1	5	0.580460	0.500000	0.0	0.0	0.391566	0.352082	0.332039	0.0	...	0.0	0.166667	0.0	0.0	0.658915	0.716377	0.011080	138	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
13091	100	194	0.781609	0.500000	0.0	0.0	0.611446	0.619359	0.566172	0.0	...	0.0	0.500000	0.0	0.0	0.395349	0.418669	0.534626	24	1	1
13092	100	195	0.436782	0.416667	0.0	0.0	0.605422	0.537388	0.671843	0.0	...	0.0	0.583333	0.0	0.0	0.333333	0.528721	0.537396	23	1	1
13093	100	196	0.465517	0.250000	0.0	0.0	0.671687	0.482014	0.414754	0.0	...	0.0	0.583333	0.0	0.0	0.372093	0.429301	0.540166	22	1	1
13094	100	197	0.281609	0.583333	0.0	0.0	0.617470	0.522128	0.626435	0.0	...	0.0	0.583333	0.0	0.0	0.403101	0.518779	0.542936	21	1	1
13095	100	198	0.574713	0.750000	0.0	0.0	0.524096	0.666667	0.721472	0.0	...	0.0	0.666667	0.0	0.0	0.434109	0.402237	0.545706	20	1	1

13096 rows × 30 columns

Modelling

In [45]:
```
# window size of 60 cycles
sequence_len = 60
```

In [46]:
```
def gen_sequence(id_data, seq_length, seq_cols):
    data_array = id_data[seq_cols].values
    num_elements = data_array.shape[0]
    for start, stop in zip(range(0, num_elements-seq_length), range(seq_length, num_elements)):
        yield data_array[start:stop, :]
```

In [47]:
```
sequence_cols = ['setting1', 'setting2', 'setting3', 'cycle_norm','s1', 's2', 's3','s4', 's5', 's6', 's7', 's8', 's9', 's10', 's11', 's12', 's13', 's14',
's15', 's16', 's17', 's18', 's19', 's20', 's21']
```

In [48]:
```
seq_gen = (list(gen_sequence(train_data[train_data['id']==id], sequence_len, sequence_cols))
           for id in train_data['id'].unique())
seq_array = np.concatenate(list(seq_gen)).astype(np.float32)
seq_array.shape
```

Out[48]: (14631, 60, 25)

In [49]:
```
def gen_labels(id_df, seq_length, label):
    data_array = id_df[label].values
    num_elements = data_array.shape[0]
    return data_array[seq_length:num_elements, :]
```

In [50]:
```
label_gen = [gen_labels(train_data[train_data['id']==id], sequence_len, ['label1'])
             for id in train_data['id'].unique()]
label_array = np.concatenate(label_gen).astype(np.float32)
label_array.shape
```

Out[50]: (14631, 1)

GRU Network

In [51]:
```
nb_features = seq_array.shape[2]
nb_out = 1
in_shape = (sequence_len, nb_features)
input_n = Input(shape=in_shape)
input_n.shape
```

Out[51]: (None, 60, 25)

In [52]:
```
# Define the sigmoid activation function
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Define the tanh activation function
def tanh(x):
    return np.tanh(x)

# Define the derivative of the tanh function
def tanh_derivative(x):
    return 1 - np.square(np.tanh(x))
```

In [53]:
```
gru1 = GRU(units=500, return_sequences=True)(input_n)
dropout1 = Dropout(0.2)(gru1)
```

In [54]:
```
gru2 = GRU(units=100, return_sequences=False)(dropout1)
dropout2 = Dropout(0.2)(gru2)
```

In [55]:
```
conv1 = Conv1D(filters=64, kernel_size=3, activation='relu')(input_n)
conv2 = Conv1D(filters=128, kernel_size=3, activation='relu')(conv1)
pool1 = MaxPooling1D(pool_size=2)(conv2)
flatten1 = Flatten()(pool1)
```

In [56]:
```
concat = concatenate([dropout2, flatten1])
```

In [57]:
```
output_n = Dense(units=nb_out, activation='sigmoid')(concat) #sigmoid (0,1) as we've used binary classification
```

In [58]:
```
#our model
model = Model(inputs=input_n, outputs=output_n)
```

In [59]:
```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [60]:
```
print(model.summary())
```

Model: "functional_3"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 60, 25)	0	-
gru_2 (GRU)	(None, 60, 500)	790,500	input_layer_1[0]…
conv1d_2 (Conv1D)	(None, 58, 64)	4,864	input_layer_1[0]…
dropout_2 (Dropout)	(None, 60, 500)	0	gru_2[0][0]
conv1d_3 (Conv1D)	(None, 56, 128)	24,704	conv1d_2[0][0]
gru_3 (GRU)	(None, 100)	180,600	dropout_2[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 28, 128)	0	conv1d_3[0][0]
dropout_3 (Dropout)	(None, 100)	0	gru_3[0][0]
flatten_1 (Flatten)	(None, 3584)	0	max_pooling1d_1[…
concatenate_1 (Concatenate)	(None, 3684)	0	dropout_3[0][0], flatten_1[0][0]
dense_1 (Dense)	(None, 1)	3,685	concatenate_1[0]…

Total params: 1,004,353 (3.83 MB)

Trainable params: 1,004,353 (3.83 MB)

Non-trainable params: 0 (0.00 B)

None

In [61]:
```
model.fit(seq_array, label_array, epochs=1, batch_size=250, validation_split=0.02, verbose=1,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0, mode='auto')])
print("Model evaluation on training data:")
print("Loss: %.2f" % scores[0])
print("Accuracy: %.2f" % scores[1])
```

58/58 ━━━━━━━━━━━━━━━━━━━━ **117s** 2s/step - accuracy: 0.8587 - loss: 0.3496 - val_accuracy: 0.9181 - val_loss: 0.1710
Model evaluation on training data:
Loss: 0.13
Accuracy: 0.95

In [62]:
```
# training metrics
scores = model.evaluate(seq_array, label_array, verbose=1, batch_size=200)
```

74/74 ━━━━━━━━━━━━━━━━━━━━ **38s** 509ms/step - accuracy: 0.9523 - loss: 0.1203

In [63]:
```
seq_array_l = [test_data[test_data['id']==id][sequence_cols].values[-sequence_len:]
               for id in test_data['id'].unique() if len(test_data[test_data['id']==id]) >= sequence_len]
seq_array_test_last = np.asarray(seq_array_l).astype(np.float32)
seq_array_l = np.asarray(seq_array_l).astype(np.float32)

# Filter labels for testing data
y_mask = [len(test_data[test_data['id']==id]) >= sequence_len for id in test_data['id'].unique()]
label_array_l = test_data.groupby('id')['label1'].nth(-1)[y_mask].values
label_array_l = label_array_l.reshape(label_array_l.shape[0],1).astype(np.float32)

print(seq_array_l.shape)
print(label_array_l.shape)
```

```
# Generate predictions for testing data
predictions = model.predict(seq_array_1)
```

```
(88, 60, 25)
(88, 1)
3/3 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step
```

In [64]:
```
scores_test = model.evaluate(seq_array_1, label_array_1, verbose=1)
print('Test Accurracy: {}'.format(scores_test[1]))
```

```
3/3 ━━━━━━━━━━━━━━━━━━━━ 1s 144ms/step - accuracy: 0.9347 - loss: 0.2029
Test Accurracy: 0.9318181872367859
```