```
import numpy as np
from numpy import savetxt
import pandas as pd
from pandas.plotting import register_matplotlib_converters
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import r2_score
import tensorflow as tf
from tensorflow import keras

%matplotlib inline
%config InlineBackend.figure_format='retina'
register_matplotlib_converters()
```

```
source = '.../Dataset.csv'
df = pd.read_csv(source, index_col=0)
df = df[['fc', 'n', 'fhy', 'load', 'ph', 'fsy', 'ds', 's', 'ps', 'dh', 'displacement']]
df.head()
```

```
train_size = int(len(df) * 0.8)
test_size = len(df) - train_size
train, test = df.iloc[0:train_size], df.iloc[train_size:len(df)]
print(len(train), len(test))
```

```
features = ['fc', 'n', 'fhy', 'ph', 'fsy', 'ds', 's', 'ps', 'dh', 'displacement']

feature_scaler = MinMaxScaler()
intensity_scaler = MinMaxScaler()

feature_scaler = feature_scaler.fit(train[features].to_numpy())
intensity_scaler = intensity_scaler.fit(train[['load']])

train.loc[:, features] = feature_scaler.transform(train[features].to_numpy())
train['load'] = intensity_scaler.transform(train[['load']])

test.loc[:, features] = feature_scaler.transform(test[features].to_numpy())
test['load'] = intensity_scaler.transform(test[['load']])
```

```
def training_data(X, y, time_steps=1):
        Xs, ys = [], []
        for i in range(len(X) - time_steps):
                v = X.iloc[i:(i + time_steps)].values
                Xs.append(v)
                ys.append(y.iloc[i + time_steps])
        return np.array(Xs), np.array(ys)
```

```
time_steps = 40 #Make predictions on the past 40 time_steps = 10h

# reshape to [samples, time_steps, n_features]

X_train, y_train = training_data(train, train.load, time_steps)
x_test, y_test = training_data(test, test.load, time_steps)

print(X_train.shape, y_train.shape)
print(x_test.shape, y_test.shape)
```

```
def model_vLSTM(units):
        model = keras.Sequential()
        #input
        model.add(keras.layers.LSTM(
                units=units,
```

```
                activation="relu",
                input_shape=(X_train.shape[1],  X_train.shape[2])
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_tLSTM_h2(units):
        model  =  keras.Sequential()
        #Input  Layer
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                input_shape=(X_train.shape[1],  X_train.shape[2])
                ))
        model.add(keras.layers.Dropout(0.2))
        #Hidden  Layer
        model.add(keras.layers.LSTM(
                units=units,
                ))
        model.add(keras.layers.Dropout(0.2))
        #Output  Layer
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_tLSTM_h3(units):
        model  =  keras.Sequential()
        #Input  Layer
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                input_shape=(X_train.shape[1],  X_train.shape[2])
                ))
        model.add(keras.layers.Dropout(0.2))
        #Hidden  Layers
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                ))
        model.add(keras.layers.Dropout(0.2))
        #Output  Layer
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_tLSTM_h4(units):
        model  =  keras.Sequential()
        #input
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                input_shape=(X_train.shape[1],  X_train.shape[2])
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                ))
        model.add(keras.layers.Dropout(0.2))
```

```
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                return_sequences=True,
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.LSTM(
                units=units,
                activation="relu",
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_BiLSTM(units):
        model  =  keras.Sequential()
        #Input  Layer
        model.add(keras.layers.Bidirectional(
                keras.layers.LSTM(
                        units=units,
                        activation="relu",
                        input_shape=(X_train.shape[1],  X_train.shape[2])
                )))
        model.add(keras.layers.Dropout(0.2))
        #Hidden  Layer
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_GRU(units):
        model  =  keras.Sequential()
        #input
        model.add(keras.layers.GRU(
                units=units,
                activation="relu",
                input_shape=(X_train.shape[1],  X_train.shape[2])
                ))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_BiGRU(units):
        model  =  keras.Sequential()
        #input
        model.add(keras.layers.Bidirectional(
                keras.layers.GRU(
                        units=units,
                        activation="relu",
                        input_shape=(X_train.shape[1],  X_train.shape[2])
                )))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  model_BiGRU_h2(units):
        model  =  keras.Sequential()
        #input
        model.add(keras.layers.Bidirectional(
                keras.layers.GRU(
                        units=units,
                        return_sequences=True,
                        activation="relu",
                        input_shape=(X_train.shape[1],  X_train.shape[2])
                )))
        model.add(keras.layers.Dropout(0.2))
        #Hidden  Layers
```

```
        model.add(keras.layers.Bidirectional(
                keras.layers.GRU(
                        units=units,
                        activation="relu",
                )))
        model.add(keras.layers.Dropout(0.2))
        model.add(keras.layers.Dense(1))
        model.compile(loss='mse',  optimizer='adam')
        return  model
```

```
def  fit_model(model):
        early_stop  =  keras.callbacks.EarlyStopping(
                monitor  =  'val_loss',
                min_delta  =  0.0,
                patience  =  10)
        history  =  model.fit(
                X_train,  y_train,
                epochs  =  100,
                validation_split  =  0.1,
                batch_size  =  64,
                shuffle  =  False,
                callbacks  =  [early_stop])
        return  history
```

```
#  Plot  train  loss  and  validation  loss
def  plot_loss  (history,  model_name):
        plt.figure(figsize  =  (8,  4))
        plt.plot(history.history['loss'],  color  =  'blue',  label='Train  Loss')
        plt.plot(history.history['val_loss'],  color  =  'red',  label='Validation  Loss')
        plt.title('Train  vs.  Validation  Loss  for  '  +  model_name)
        plt.ylabel('Loss')
        plt.xlabel('epoch')
        plt.legend(loc='upper  right')
        plt.savefig('G:/Failure  mode  prediction/Cycling  column/Method2/'+'Loss_'+model_name+'.jpg',  format='
```

```
def  save_model(model,  model_name):
        model.save('G:/Failure  mode  prediction/Cycling  column/Method2/'+model_name)
        return  None
```

```
def  prediction(model,  x_test,  y_test):
        y_pred  =  model.predict(x_test)
        y_train_inv  =  intensity_scaler.inverse_transform(y_train.reshape(1,  -1))
        y_test_inv  =  intensity_scaler.inverse_transform(y_test.reshape(1,  -1))
        y_pred_inv  =  intensity_scaler.inverse_transform(y_pred)
        return  y_pred,  y_pred_inv,  y_train_inv,  y_test_inv
```

```
def  plot_forecast(prediction_model,  y_test,  model_name):
        y_pred,  y_pred_inv,  y_train_inv,  y_test_inv  =  prediction_model
        plt.plot(np.arange(0,  len(y_train)),  y_train_inv.flatten(),  'g',  label="history")
        plt.plot(np.arange(len(y_train),  len(y_train)  +  len(y_test)),  y_test_inv.flatten(),  marker='.',  lab
        plt.plot(np.arange(len(y_train),  len(y_train)  +  len(y_test)),  y_pred_inv.flatten(),  'r',  label="Pre
        plt.ylabel('Load  (kN)')
        plt.xlabel('Time  Step  (s)')
        plt.title('Prediction  of  applied  load  '+  model_name)
        plt.legend()
        plt.savefig('G:/Failure  mode  prediction/Cycling  column/Method2/'+'Forecast_'+model_name+'.jpg',  forma
        plt.show();
```

```
def  plot_compare(prediction_model  ,y_test,  model_name):

        y_pred,  y_pred_inv,  y_train_inv,  y_test_inv  =  prediction_model
        plt.plot(y_test_inv.flatten(),  marker='.',  label="Test  data")
        plt.plot(y_pred_inv.flatten(),  'r',  label="Prediction  "+model_name)
        plt.ylabel('Load  (kN)')
        plt.xlabel('Time  Step  (s)')
        plt.title('Prediction  of  applied  load  '+model_name)
        plt.legend()
```

```
        plt.savefig('G:/Failure mode prediction/Cycling column/Method2/'+'TestPrediction_'+model_name+'.jpg',
        plt.show();
```

```
def plot_compare_models_2(prediction_model_1, prediction_model_2 ,y_test, model_name_1, model_name_2):

        y_pred_1, y_pred_inv_1, y_train_inv_1, y_test_inv_1 = prediction_model_1
        y_pred_2, y_pred_inv_2, y_train_inv_2, y_test_inv_2 = prediction_model_2

        plt.plot(y_test_inv_1.flatten(), marker='.', label="Test data")
        plt.plot(y_pred_inv_1.flatten(), 'r', label="Prediction "+model_name_1)
        plt.plot(y_pred_inv_2.flatten(), 'g', label="Prediction "+model_name_2)
        plt.ylabel('Load (kN)')
        plt.xlabel('Time Step (s)')
        plt.title('Test data vs Prediction data: '+model_name_1+' vs. '+model_name_2)
        plt.legend()
        plt.savefig('G:/Failure mode prediction/Cycling column/Method2/'+'TestPrediction_'+model_name_1+'vs'+m
        plt.show();
```

```
def plot_compare_models_3(prediction_model_1, prediction_model_2, prediction_model_3, y_test, model_name_1,

        y_pred_1, y_pred_inv_1, y_train_inv_1, y_test_inv_1 = prediction_model_1
        y_pred_2, y_pred_inv_2, y_train_inv_2, y_test_inv_2 = prediction_model_2
        y_pred_3, y_pred_inv_3, y_train_inv_3, y_test_inv_3 = prediction_model_3

        plt.plot(y_test_inv_1.flatten(), marker='.', label="Test data")
        plt.plot(y_pred_inv_1.flatten(), 'r', label="Prediction "+model_name_1)
        plt.plot(y_pred_inv_2.flatten(), 'g', label="Prediction "+model_name_2)
        plt.plot(y_pred_inv_3.flatten(), 'gray', label="Prediction "+model_name_3)

        plt.ylabel('Load (kN)')
        plt.xlabel('Time Step (s)')
        plt.title('Test data vs Prediction data: '+model_name_1+' vs. '+model_name_2 + ' vs. '+model_r
        plt.legend()
        plt.savefig('G:/Failure mode prediction/Cycling column/Method2/'+'TestPrediction_'+model_name_1+'vs'+m
        plt.show();
```

```
def evaluate_performance(prediction_model, y_test, model_name):
        y_pred, y_pred_inv, y_train_inv, y_test_inv = prediction_model

        y_test_inv = y_test_inv.reshape(-1,1)
        errors = y_pred_inv - y_test_inv


        mae = round(np.abs(errors).mean(),2)
        mse = round(np.square(errors).mean(),2)
        rmse = round(np.sqrt(mse),2)
        r_sq = round(r2_score(y_test, y_pred),2)

        performance = pd.DataFrame({"Model": [model_name], "MAE": [mae], "MSE": [mse], "RMSE": [mse], '
        performance.to_csv('G:/Failure mode prediction/Cycling column/Method2/'+'Measures_'+model_name+'.csv')

        print('Performance of '+model_name+ ':')
        print('MAE:\t', mae)
        print('MSE:\t', mse)
        print('RMSE:\t', rmse)
        print('R^2:\t', r_sq)
        return mae, mse, rmse, r_sq
```

```
vLSTM_n4 = model_vLSTM(4)
vLSTM_n8 = model_vLSTM(8)
vLSTM_n16 = model_vLSTM(16)
vLSTM_n32 = model_vLSTM(32)
```

```
# LSTM models with 32 hidden neurons.
tLSTM_n32h2 = model_tLSTM_h2(32)
tLSTM_n32h3 = model_tLSTM_h3(32)
tLSTM_n32h4 = model_tLSTM_h4(32)
GRU_n32 = model_GRU(32)
```

```
BiLSTM_n32 = model_BiLSTM(32)
BiGRU_n32 = model_BiGRU(32)
```

```
history_vLSTM_n4 = fit_model(vLSTM_n4)
history_vLSTM_n8 = fit_model(vLSTM_n8)
history_vLSTM_n16 = fit_model(vLSTM_n16)
history_vLSTM_n32 = fit_model(vLSTM_n32)
```

```
history_tLSTM_n32h2 = fit_model(tLSTM_n32h2)
history_tLSTM_n32h3 = fit_model(tLSTM_n32h3)
history_tLSTM_n32h4 = fit_model(tLSTM_n32h4)
history_GRU_n32 = fit_model(GRU_n32)
history_BiLSTM_n32 = fit_model(BiLSTM_n32)
history_BiGRU_n32 = fit_model(BiGRU_n32)
```

```
plot_loss(history_vLSTM_n4, 'vLSTM_n4')
plot_loss(history_vLSTM_n8, 'vLSTM_n8')
plot_loss(history_vLSTM_n16, 'vLSTM_n16')
plot_loss(history_vLSTM_n32, 'vLSTM_n32')
```

```
plot_loss(history_tLSTM_n32h2, 'tLSTM_n32h2')
plot_loss(history_tLSTM_n32h3, 'tLSTM_n32h3')
plot_loss(history_tLSTM_n32h4, 'tLSTM_n32h4')
plot_loss(history_GRU_n32, 'GRU_n32')
plot_loss(history_BiLSTM_n32, 'BiLSTM_n32')
plot_loss(history_BiGRU_n32, 'BiGRU_n32')
```

```
# Calculate predictions neurons
prediction_vLSTM_n4 = prediction(vLSTM_n4, x_test, y_test)
prediction_vLSTM_n8 = prediction(vLSTM_n8, x_test, y_test)
prediction_vLSTM_n16 = prediction(vLSTM_n16, x_test, y_test)
prediction_vLSTM_n32 = prediction(vLSTM_n32, x_test, y_test)
```

```
# Calculate predictions layers
prediction_tLSTM_n32h2 = prediction(tLSTM_n32h2, x_test, y_test)
prediction_tLSTM_n32h3 = prediction(tLSTM_n32h3, x_test, y_test)
prediction_tLSTM_n32h4 = prediction(tLSTM_n32h4, x_test, y_test)
prediction_GRU_n32 = prediction(GRU_n32, x_test, y_test)
prediction_BiLSTM_n32 = prediction(BiLSTM_n32, x_test, y_test)
prediction_BiGRU_n32 = prediction(BiGRU_n32, x_test, y_test)
```

```
plot_forecast(prediction_vLSTM_n4, y_test, 'vLSTM_n4')
plot_forecast(prediction_vLSTM_n8, y_test, 'vLSTM_n8')
plot_forecast(prediction_vLSTM_n16, y_test, 'vLSTM_n16')
plot_forecast(prediction_vLSTM_n32, y_test, 'vLSTM_n32')
```

```
plot_forecast(prediction_tLSTM_n32h2, y_test, 'tLSTM_n32h2')
plot_forecast(prediction_tLSTM_n32h3, y_test, 'tLSTM_n32h3')
plot_forecast(prediction_tLSTM_n32h4, y_test, 'tLSTM_n32h4')
plot_forecast(prediction_GRU_n32, y_test, 'GRU_n32')
plot_forecast(prediction_BiLSTM_n32, y_test, 'BiLSTM_n32')
plot_forecast(prediction_BiGRU_n32, y_test, 'BiGRU_n32')
```

```
plot_compare(prediction_vLSTM_n4, y_test, 'vLSTM_n4')
plot_compare(prediction_vLSTM_n8, y_test, 'vLSTM_n8')
plot_compare(prediction_vLSTM_n16, y_test, 'vLSTM_n16')
plot_compare(prediction_vLSTM_n32, y_test, 'vLSTM_n32')
```

```
plot_compare(prediction_tLSTM_n32h2, y_test, 'tLSTM_n32h2')
plot_compare(prediction_tLSTM_n32h3, y_test, 'tLSTM_n32h3')
plot_compare(prediction_tLSTM_n32h4, y_test, 'tLSTM_n32h4')
plot_compare(prediction_GRU_n32, y_test, 'GRU_n32')
```

```
plot_compare(prediction_BiLSTM_n32,  y_test,  'BiLSTM_n32')
plot_compare(prediction_BiGRU_n32,  y_test,  'BiGRU_n32')
```

```
evaluate_performance(prediction_vLSTM_n4,  y_test,  'vLSTM_n4')
evaluate_performance(prediction_vLSTM_n8,  y_test,  'vLSTM_n8')
evaluate_performance(prediction_vLSTM_n16,  y_test,  'vLSTM_n16')
evaluate_performance(prediction_vLSTM_n32,  y_test,  'vLSTM_n32')
```

```
evaluate_performance(prediction_tLSTM_n32h2,  y_test,  'tLSTM_n32h2')
evaluate_performance(prediction_tLSTM_n32h3,  y_test,  'tLSTM_n32h3')
evaluate_performance(prediction_tLSTM_n32h4,  y_test,  'tLSTM_n32h4')
evaluate_performance(prediction_GRU_n32,  y_test,  'GRU_n32')
evaluate_performance(prediction_BiLSTM_n32,  y_test,  'BiLSTM_n32')
evaluate_performance(prediction_BiGRU_n32,  y_test,  'BiGRU_n32')
```

```
dataframe  =  pd.DataFrame({'y_pred_inv':prediction_tLSTM_n32h2})
dataframe.to_csv("outputn32h2.csv",index=True,sep=',')
```