from pandas import DataFrame

from pandas import Series

from pandas import concat

from pandas import read\_csv

from pandas import datetime

from sklearn.metrics import median\_absolute\_error

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from math import sqrt

from matplotlib import pyplot

import numpy as np

import pandas as pd

# Hardcode all variables

predict\_batch\_size\_exp = 1

epoch\_exp = 50

neurons\_exp = 5

predict\_values\_exp = 168

lag\_exp=24

# frame a sequence as a supervised learning problem

def timeseries\_to\_supervised(data, lag=1):

df = DataFrame(data)

columns = [df.shift(i) for i in range(1, lag+1)]

columns.append(df)

df = concat(columns, axis=1)

df.fillna(0, inplace=True)

return df

# create a differenced series

def difference(dataset, interval=1):

diff = list()

for i in range(interval, len(dataset)):

value = dataset[i] - dataset[i - interval]

diff.append(value)

return Series(diff)

#In [5]:

# invert differenced value

def inverse\_difference(history, yhat, interval=1):

return yhat + history[-interval]

#In [6]:

# scale train and test data to [-1, 1]

def scale(train, test):

# fit scaler

scaler = MinMaxScaler(feature\_range=(-1, 1))

scaler = scaler.fit(train)

# transform train

train = train.reshape(train.shape[0], train.shape[1])

train\_scaled = scaler.transform(train)

# transform test

test = test.reshape(test.shape[0], test.shape[1])

test\_scaled = scaler.transform(test)

return scaler, train\_scaled, test\_scaled

#In [7]:

# inverse scaling for a forecasted value

def invert\_scale(scaler, X, value):

new\_row = [x for x in X] + [value]

array = np.array(new\_row)

array = array.reshape(1, len(array))

inverted = scaler.inverse\_transform(array)

return inverted[0, -1]

#In [8]:

# fit an LSTM network to training data

from keras.layers import Activation, Dense, BatchNormalization, TimeDistributed

def fit\_lstm(train, batch\_size, nb\_epoch, neurons):

X, y = train[:, 0:-1], train[:, -1]

X = X.reshape(X.shape[0], 1, X.shape[1])

model = Sequential()

model.add(LSTM(neurons\_exp, dropout = 0.1 ,batch\_input\_shape=(batch\_size, X.shape[1], X.shape[2]), stateful=True))

model.add(BatchNormalization())

model.add(Dense(50))

model.add(Activation('relu'))

model.add(Dense(50))

model.add(Activation('tanh'))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error', optimizer='adam')

for i in range(nb\_epoch):

model.fit(X, y, epochs=1, batch\_size=batch\_size, verbose=1, shuffle=False)

model.reset\_states()

return model

#In [9]:

# make a one-step forecast with new model with batch size 1

def forecast\_lstm(model, batch\_size, X):

X = X.reshape(1, 1, len(X))

# re-define model

new\_model = Sequential()

new\_model.add(LSTM(neurons\_exp, dropout = 0.1 , batch\_input\_shape=(batch\_size, X.shape[1], X.shape[2]), stateful=True))

new\_model.add(BatchNormalization())

new\_model.add(Dense(50))

new\_model.add(Activation('relu'))

new\_model.add(Dense(50))

new\_model.add(Activation('tanh'))

new\_model.add(Dense(1))

# copy weights

old\_weights = model.get\_weights()

new\_model.set\_weights(old\_weights)

# compile model

new\_model.compile(loss='mean\_squared\_error', optimizer='adam')

#print(X)

yhat = new\_model.predict(X, batch\_size=1)

return yhat[0,0]

#In [10]:

''' Loading data '''

import pandas as pd

series = pd.read\_excel('hi.xlsx',index\_col="DateTime")

series.head()

'''Drop all the features as we will not be having any in production'''

del series['Air temperature | (\'C)']

del series['Pressure | (atm)']

del series['Wind speed | (m/s)']

del series['Wind direction | (deg)']

series.head()

for i in range(0,16):

series = series[:-1]

series.tail()

# transform data to be stationary

raw\_values = series.values

diff\_values = difference(raw\_values, 1)

#In [14]:

# transform data to be supervised learning

supervised = timeseries\_to\_supervised(diff\_values, lag\_exp)

supervised\_values = supervised.values

#In [15]:

# split data into train and test-sets

train, test = supervised\_values[0:-predict\_values\_exp], supervised\_values[-predict\_values\_exp:]

#In [16]:

# transform the scale of the data

scaler, train\_scaled, test\_scaled = scale(train, test)

# fit the model

fit\_batch\_size\_exp = 419

lstm\_model = fit\_lstm(train\_scaled, fit\_batch\_size\_exp, epoch\_exp, neurons\_exp)

# walk-forward validation on the test data

predictions = list()

expectations = list()

test\_pred = list()

for i in range(len(test\_scaled)):

# make one-step forecast

X, y = test\_scaled[i, 0:-1], test\_scaled[i, -1]

yhat = forecast\_lstm(lstm\_model, 1, X)#batch\_size\_exp to 1

'''# Start Debug prints

print("X: %", X)

print("yhat: %", yhat)

# End Debug prints'''

# Replacing value in test scaled with the predicted value.

test\_pred = [yhat] + test\_pred

if len(test\_pred) > lag\_exp+1:

test\_pred = test\_pred[:-1]

if i+1<len(test\_scaled):

if i+1 > lag\_exp+1:

test\_scaled[i+1] = test\_pred

else:

test\_scaled[i+1] = np.concatenate((test\_pred, test\_scaled[i+1, i+1:]),axis=0)

# invert scaling

yhat = invert\_scale(scaler, X, yhat)

# invert differencing

yhat = inverse\_difference(raw\_values, yhat, len(test\_scaled)+1-i)

# store forecast

expected = raw\_values[len(train) + i + 1]

if expected != 0:

predictions.append(yhat)

expectations.append(expected)

print('Hour=%d, Predicted=%f, Expected=%f' % (i+1, yhat, expected))

expectations = np.array(expectations)

predictions = np.array(predictions)

print("Mean Absolute Percent Error: ", (np.mean(np.abs((expectations - predictions) / expectations))\*100))

# line plot of observed vs predicted

pyplot.figure(figsize=(20,8))

pyplot.plot(expectations, label="True")

pyplot.plot(predictions, label="Predicted")

pyplot.legend(loc='upper right')

pyplot.xlabel("Number of hours")

pyplot.ylabel("Power generated by system (kW)")

pyplot.show()