```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Oct. 7, 2019

@author: Brice Loose

Script for LSTM fit to SWIMS data.

Ref:  Loose et al., (2020), Instrument bias correction with machine learning
algorithms: Application to field-portable mass spectrometry, Frontiers in
Geoscience, DOI: ??

.
#
# DISCLAIMER:
#    This is provided "as is" without warranty of any kind.
#==========================================================================

"""



## Import Libraries ##
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
# multivariate multi-step encoder-decoder lstm
from numpy import split
from numpy import array
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.layers import Dropout



# split a univariate dataset into train/test sets
def split_dataset(data,n_input):
    # split into standard weeks
    train, test = data[0:-n_input*2], data[-n_input*2:]
    # restructure into windows of weekly data
    train = array(split(train, len(train)/n_input))
    test = array(split(test, len(test)/n_input))
    return train, test


# convert history into inputs and outputs
def to_supervised(train, n_input, n_out=100):
    # flatten data
    data = train.reshape((train.shape[0]*train.shape[1], train.shape[2]))
    X, y = list(), list()
    in_start = 0
    # step over the entire history one time step at a time
    for _ in range(len(data)):
        # define the end of the input sequence
        in_end = in_start + n_input
        #out_end = in_end + n_out
        # ensure we have enough data for this instance
        if in_end <= len(data)-1:
            X.append(data[in_start:in_end, 0:])
            y.append(data[in_start:in_end, -1])

        # move along one time step
        in_start += 1
    return array(X), array(y)



# Set pre-determined values of hyperparameters
verbose, epochs, batch_size = 1, 20, 80
n_input = 100
# Index for the target data (O2)
target = 64
# Indices for the environmental covariates.
idx = [62,5,36,28,57,3,64]


#%%  TRAINING STEP HERE

# Laad the file
ms = pd.read_csv(r'Insitu_cal.csv',header='infer',index_col=0);
hdrs = ms.columns.values

# Clean up nans and missing data
X = ms[hdrs[idx]].interpolate(method='linear', order=1, limit_direction='both')
y15 = ms[hdrs[target]]

X_train = X
y_train = y15


# Standardize the environmental covariates to ensure equal weighting.
Xf = X.values
Xf_mean = Xf.mean(axis=0)
Xf_std = Xf.std(axis=0)
x = (Xf-Xf_mean)/Xf_std

#shorten dataset to split into pieces
x = x[0:len(Xf)-np.mod(len(Xf),n_input)]
```

```python
train, test = split_dataset(x,n_input)
test_x, test_y = to_supervised(test, n_input)
train_x, train_y = to_supervised(train, n_input)

# Extract tensor dimensions
n_timesteps, n_features, n_outputs = train_x.shape[1], train_x.shape[2],
 train_y.shape[1]
    # reshape output into [samples, timesteps, features]
train_y = train_y.reshape((train_y.shape[0], train_y.shape[1], 1))

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

    # define model
model = Sequential()
model.add(LSTM(50, activation='tanh', input_shape=(n_timesteps, n_features)))
model.add(RepeatVector(n_outputs))
model.add(Dropout(0.4))
model.add(LSTM(50, activation='tanh', return_sequences=True))
model.add(TimeDistributed(Dense(20, activation='relu')))
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')


    # fit network
result = model.fit(train_x, train_y, epochs=epochs, batch_size=batch_size,
 verbose=verbose,
   validation_data=(test_x, test_y))
model15=model

train_x, train_y = to_supervised(train, n_input)


#  Get reconstructed target data
yhat = model15.predict(train_x,verbose=1)
#  Undo standardization.
mz = yhat*Xf_std[-1]+Xf_mean[-1]

#%%
###------------- NNLS optimization function for calib -------------- #####
from scipy.optimize import minimize
def optocf(x0,ytrain,yp,o2):
    import Solubility as s
    import numpy as np
    den = (np.mean(ytrain-x0))
    CF = 210.68/den
    yrect = (yp + x0)*CF

    return np.mean(np.abs(o2 - yrect))

# Make data set with progressive change in CF to cover changes in the
 calibration
```

```
y = y15; lstm = model15


MA = pd.read_csv(r'SWIMS_Lat36_38.csv',header='infer',index_col=0); hdrs =
 MA.columns.values


#%%     PREDICTION STEP HERE


X = MA[hdrs[idx]].interpolate(method='linear', order=1, limit_direction='both')

# Standardize the environmental covariates to ensure equal weighting.
Xf = X.values
Xf_mean = Xf.mean(axis=0)
Xf_std = Xf.std(axis=0)
x = (Xf-Xf_mean)/Xf_std

pad = n_input-np.mod(len(x),n_input)
x = np.append(x,np.ones([pad+n_input,len(idx)]),axis=0)

trane = array(split(x, len(x)/n_input))
trane_x, trane_y = to_supervised(trane, n_input)


# Apply LSTM model to predict bias signal in SWIMS data
yhat = lstm.predict(trane_x,verbose=1)


# Reconstruct target data using standard normal
yr = pd.DataFrame(yhat[0:-pad-1,1,:])*MA[hdrs[target]].std()
 +MA[hdrs[target]].mean()


# Merge reconstructed target with the rest of the data
MA = pd.concat([MA,yr],axis=1).dropna(how='any')


####  Calibration step.  ################
### Non-linear optimization to find best fit to SBE oxygen data #
## See Section 2.3 on calibration after bias removal ###
t = minimize(optocf,1e-15,args=(y,MA[hdrs[target]]-MA[0],MA[hdrs[8]]),
             method='Nelder-Mead',tol=1e-13)

# This is y-intercept in calib.
ic = t.x

# Equil. solubility at S=35.5 and T =23 C.
cf = 210.68/np.mean(y-ic)
```

```
yraw = yr; yraw.rename(columns={0:'yraw'},inplace=True)

# Apply calibration
yrect = pd.DataFrame((MA[hdrs[target]]-yr.iloc[:,0]+ic)*cf,columns={'yrect'})

# Merge with the rest of the dataframe
Yp = pd.concat([MA,yraw,yrect],axis=1)


#%%  Plotting section

mse = np.round(np.sqrt(np.nansum((Yp['yrect'] - Yp[hdrs[8]])**2)/
 len(Yp[hdrs[8]]))),3)


f,ax = plt.subplots(1,1)
ax.plot(Yp['yrect'],'r',label='SWIMS')
ax.plot(Yp[hdrs[8]],'g',label='SBE35DO')
ax.legend()
ax.set_ylim(100,300)
ax.set_title('RMSE='+mse.astype(str))
plt.show()


plt.figure()
plt.plot(result.history['loss'],label='loss')
plt.plot(result.history['val_loss'],label='valid_loss')
plt.xlabel('epoch');
plt.legend()
plt.show()
```