DLM_basic2_HyperParamTuning

February 28, 2019

In [1]:
```
#%% IMPORT PACKAGES
from keras.layers import Input, Dense, Dropout
from keras.models import Model
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
import os
import time
import h5py
import pickle

#from matplotlib import rcParams  # next 3 lines set font family for plotting
#rcParams['font.family'] = 'serif'
#rcParams['font.sans-serif'] = ['TImes New Roman']
#import matplotlib.pyplot as plt
#
#from keras.utils import plot_model
#os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/Graphviz2.38/release/bin'

# set working directory (change the following path to match your directory structure)
main = 'C:\\Users\\Kathy_Breen\\Documents\\DL_Seminar\\Week3'  # set directory path where this f
os.chdir(main)  # make sure the Spyder is pointing to the correct folder
```

```
Using TensorFlow backend.
```

In [2]:
```
#%% SETTINGS FOR REPRODUCIBLE RESULTS DURING DEVELOPMENT

import tensorflow as tf
import random as rn

# The below is necessary in Python 3.2.3 onwards to
# have reproducible behavior for certain hash-based operations.
# See these references for further details:
```

```
# https://docs.python.org/3.4/using/cmdline.html#envvar-PYTHONHASHSEED
# https://github.com/keras-team/keras/issues/2280#issuecomment-306959926

#import os
os.environ['PYTHONHASHSEED'] = '0'

# The below is necessary for starting Numpy generated random numbers
# in a well-defined initial state.

np.random.seed(42)

# The below is necessary for starting core Python generated random numbers
# in a well-defined state.

rn.seed(12345)

# Force TensorFlow to use single thread.
# Multiple threads are a potential source of
# non-reproducible results.
# For further details, see: https://stackoverflow.com/questions/42022950/which-seeds-have-to-be-

session_conf = tf.ConfigProto(intra_op_parallelism_threads=1, inter_op_parallelism_threads=1)

from keras import backend as K

# The below tf.set_random_seed() will make random number generation
# in the TensorFlow backend have a well-defined initial state.
# For further details, see: https://www.tensorflow.org/api_docs/python/tf/set_random_seed

tf.set_random_seed(1234)

sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
K.set_session(sess)
```

In [3]:

```
#%% Read in *.hdf5 data sets

with h5py.File('X.hdf5','r') as f:
    X_train = np.array(f["X_train"])
    X_test = np.array(f["X_test"])

with h5py.File('Y.hdf5','r') as f:
    Y_train = np.array(f["Y_train"])
    Y_test = np.array(f["Y_test"])
```

In [4]:

```
#%% Variables
```

```
tic = time.time()  # start a timer

# define number of X and Y features
X_Nfeatures = X_train.shape[1]
Y_Nfeatures = Y_train.shape[1]
epochs = 5000
```

In [5]:

```
#%% Build MLP model inside a function to call later

def run_MLP(X_Nfeatures, Y_Nfeatures, X_train, X_test, Y_train, Y_test, epochs, batch_size,
do, Nlyr, Nnodes, run):
    # create input layer..........
    main_input = Input(shape=(X_Nfeatures),
                       dtype='float',
                       batch_shape=(batch_size,X_Nfeatures),
                       name='main_input'
                       )
    #create hidden layer..........
    hidden_layer = Dense(Nnodes, activation='relu', name='hidden_layer1')(main_input)
    # add dropout to hidden layer
    Dropout(do)(hidden_layer)
    if Nlyr > 1:
        for i in range(1,Nlyr):
            hidden_layer = Dense(Nnodes, activation='relu',
                name='hidden_layer'+str(i+1))(hidden_layer)
            # add dropout to hidden layer
            Dropout(do)(hidden_layer)

    # create output layer
    main_output = Dense(Y_Nfeatures, name='main_output')(hidden_layer)

    # feed datasets into model for training
    model = Model(inputs=[main_input],
                  outputs=[main_output]
                  )

    # compile the model with desired configuration
    model.compile(loss='mean_squared_error',
                  optimizer='adagrad',
                  metrics=['mae'])

    early_stop = EarlyStopping(monitor='val_loss', # quantity to monitor
                               min_delta=0.0001,  # min change to qualify as an improvement
                               patience=100, # stop after #epochs with no improvement
                               verbose=1)  # print messages
    reduce_lr = ReduceLROnPlateau(monitor='val_loss',
                                  factor=0.2,  # reduction factor (new_lr = lr * factor)
                                  patience=15,
                                  verbose=1)

    # train the model, and store training information in the history object
    history = model.fit([X_train],[Y_train],
                        epochs=epochs,
                        batch_size = batch_size,
                        validation_data=([X_test], [Y_test]),
#                        callbacks=[reduce_lr]
#                        callbacks=[early_stop]
                        callbacks=[reduce_lr,early_stop]
                        )
    histdict = history.history

    predict = model.predict([X_test],batch_size=batch_size)
    Y_mse = mean_squared_error(predict,Y_test)

#    main_loss_train = histdict['loss']
#    main_loss_test = histdict['val_loss']
#    xplot = list(range(len(main_loss_train)))
#
#    fig = plt.figure(num=1, figsize=(8,6))
#    ax = fig.add_subplot(111)
#    train = ax.plot(xplot,main_loss_train,'b-',label='Train',linewidth=4)
#    test = ax.plot(xplot,main_loss_test,'r-',label='Test',linewidth=4)
#    ax.set_xlabel('Epochs')
#    ax.set_ylabel('Loss')
#    curves = train+test
#    labels = [c.get_label() for c in curves]
#    ax.legend(curves, labels, loc=0)
#    plt.tight_layout()
#    plt.title(str(hparams.loc[run,:]))
#    plt.savefig('main_loss_' + str(run) + '.jpg')
#    plt.show()
#
#    fig = plt.figure(num=2, figsize=(8,6))
#    ax = fig.add_subplot(111)
#    y_true = ax.plot(X_test,Y_test,'ko',markersize=16,label=r'$Y_{true}$')
#    y_pred = ax.plot(X_test,predict,'*',color='#009191',markersize=10,label=r'$\hat{Y}_{main}$'
#    ax.set_xlabel(r'$X$')
#    ax.set_ylabel(r'$Y$')
#    curves = y_true+y_pred
#    labels = [c.get_label() for c in curves]
#    ax.legend(curves, labels, loc=0)
#    plt.tight_layout()
#    plt.title(str(hparams.loc[run,:]))
#    plt.savefig('ypred2_' + str(run) + '.jpg')
#    plt.show()
#
#    plot_model(model, to_file='DLM_basic2_structure_' + str(run) +'.jpg', show_shapes=True, sho

    return histdict, predict, Y_mse
```

In [6]:

```
#%% Call the function to perform hyperparaeter tuning

# run model with hyperparameter values and save RMSE and loss to a dictionary
rundict = {}

tic = time.time()
hparams = pd.read_csv('h_tuning.txt',sep='\t')
for row in range(hparams.shape[0]):
    batch_size = hparams.loc[row,'batch_size']
    do = hparams.loc[row,'do']
    Nlyr = hparams.loc[row,'Nlyr']
    Nnodes = hparams.loc[row,'Nnodes']

    # MLP
    histdict, predict, Y_mse = run_MLP(X_Nfeatures, Y_Nfeatures, X_train, X_test, Y_train,
        Y_test, epochs, batch_size, do, Nlyr, Nnodes, row)
    rundict[row] = {'histdict': histdict, 'predict': predict, 'Y_mse': Y_mse}

toc = (time.time() - tic)/60
print('Elapsed time: ' + str(toc) + ' minutes')

pickle.dump(rundict, open( "HyperParamOut.pkl", "wb" ))
```