Projet 4MA - LSTM

In [ ]:
```
# Imports
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import linregress
import sklearn
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler , scale
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.cluster import KMeans
from matplotlib import colors
import pylab as pyl
import math

import pywt
import scipy.io as sio
import pandas as pd
import holoviews as hv
import param
import panel as pn
from panel.pane import LaTeX
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.preprocessing.sequence import TimeseriesGenerator


from PIL import Image
from io import BytesIO
import requests
import warnings
from ipywidgets import interact
hv.extension('bokeh')

import tensorflow
from tensorflow import keras
#from plot_keras_history import plot_history
import random
from keras import backend as K
import tensorflow_probability as tfp

scaler = StandardScaler()
```

Partie 0 : Préparation des données

Faire un notebook pour cette partie ? Puis l'import dans les gros notebooks ? (comme mikael)

In [ ]:
```
# Chargement du dataset
data_pepsi = pd.read_csv("../databases/PEPSIR_raw_LF_perfect.csv", sep=";")
print('Pepsi -> Missing Data : ',data_pepsi.isna().sum().sum(), '  Shape is : ', dat
display(data_pepsi)
```

In [ ]:
```
### Modification des valeurs de alpha et beta ( avant suppression de Q et W) ###


#calculate new alpha&beta
A = np.zeros(data_pepsi.shape[0])
B = np.zeros(data_pepsi.shape[0])
cont = 0
```

```
data_pepsi_aux = data_pepsi.groupby("river", as_index=False).mean()
rivers = data_pepsi_aux['river'].tolist()
for i in data_pepsi["river"].unique() :
    river_data = data_pepsi[data_pepsi["river"] == i]
    aux = river_data.groupby("reach", as_index=False).mean()
    reachs = aux['reach'].tolist()
    for j in reachs:
        reach_data = river_data[river_data["reach"] == j]

        A0r = reach_data["A0"]
        dArp = reach_data["dA"]
        Qrp = reach_data["Q"]
        Srp = reach_data["S"]
        Wrp = reach_data["W"]
        Zrp = reach_data["height"]
        Zr0 = np.min(Zrp)
        Wr0 = np.min(Wrp)
        c1rp = Wrp**(-2./5.) * Srp**(3./10.)
        c2rp = c1rp * dArp
        c3rp = (Zrp - Zr0) #à revoir potentiellement, cf. Kévin
        c4r = 1.0 / Wr0

        x = c4r * A0r + c3rp
        y = Qrp**(3./5.) / (c1rp * A0r + c2rp)
        res = linregress(np.log(x), np.log(y))

        a = res.slope
        b = res.intercept
        alpha = np.exp(b)**(5./3.)
        beta = a * 5. / 3.

        if math.isnan(alpha) or math.isnan(beta):
                alpha = data_pepsi["alpha"].loc[cont]
                beta  = data_pepsi["beta"].loc[cont]


        A[cont :cont + reach_data.shape[0]] = alpha
        B[cont :cont + reach_data.shape[0]] = beta
        cont += reach_data.shape[0]
```

In [ ]:
```
#add new alpha&beta to the original dataframe
data_pepsi=data_pepsi.drop(columns=['alpha','beta'])
data_pepsi.reset_index(drop=True, inplace=True)
data_pepsi["alpha"] = pd.DataFrame(A)
data_pepsi["beta"] = pd.DataFrame(B)
```

In [ ]:
```
# Suppression des débits > 100 et W < 80
data_pepsi = data_pepsi.loc[data_pepsi['Q']>100]
data_pepsi = data_pepsi.loc[data_pepsi['W']> 80]
```

In [ ]:
```
river_means_pepsi=data_pepsi.groupby("river", as_index=False).mean()
```

Série temporelle avec moyenne sur les reach

In [ ]:
```
data_appr = pd.DataFrame(data_pepsi)
# for river_name in data_pepsi.river.unique():

#     data_temp = data_pepsi[data_pepsi.river.isin([river_name])].copy()
#     #print("Shape data_temp for ", river_name ," : " ,data_temp.shape)


#     data_mean_temp = data_temp.groupby("day", as_index=False).mean().copy()
```

```
#     data_mean_temp.insert(0,'river', river_name)

#     #print(" => Shape data_mean : " , data_mean_temp.shape)

#     data_appr = data_appr.append(data_mean_temp, ignore_index=True).drop('reach',a

data_appr = data_appr.set_index('river').copy()
data_appr = data_appr[data_appr["reach"] ==1]
#print("Shape après moyenne sur chaque reach : " , data_appr.shape)
```

In [ ]:
```
for i in data_appr.index.unique():
    print(i, " : ",  (data_appr.loc[[i]].day.count()))
```

Sélection d'une rivière pour le LSTM

In [ ]:
```
river_name = "MissouriDownstream"

data_river = data_appr.loc[[river_name]].copy()

#data_river.sort_values(['day'],  inplace=True) # si besoin de trier dans l'ordre ch
data_river.reset_index(drop=True, inplace=True)

print(" Shape : " , data_river.shape)
```

In [ ]:
```
data_river["Q"].plot()
plt.axhline(data_river["Q"][0],linestyle = "--",color='red',label = "1st day value")
plt.grid()
plt.xlabel("Days")
plt.ylabel("Q (m3/s)")
plt.xlim(0,450)
plt.legend()

plt.show()
```

In [ ]:
```
def pt_cont(data,deb,fin,tol = 10):
    mini = min(np.abs(data["Q"][deb:fin] - data["Q"][0]))
    while tol < mini:
        if deb > 200:
            deb -=5
            fin +=5
            mini = min(np.abs(data["Q"][deb:fin] - data["Q"][0]))
    return deb + np.argmin(np.abs(data["Q"][deb:fin] - data["Q"][0]))
```

In [ ]:
```
# On réduit le data set à une année environ avec continuité entre 1er jour et dernie
data_river = data_river.loc[:pt_cont(data_river,350,370)]

print(" New shape : " , data_river.shape)
```

In [ ]:
```
c = data_river.corr()
c.style.background_gradient(cmap='coolwarm')
```

In [ ]:
```
# On périodise le dataset
nb_p = 5
data_riverP = pd.concat([data_river]*nb_p, ignore_index = True)

print("New shape :" , data_riverP.shape)
data_riverP.head()
```

In [ ]:
```
columns_used = ['day','river','reach','W','dA','S','Q']
to_drop = data_riverP.columns.difference(columns_used)
data_riverP.drop( to_drop, axis=1, inplace=True)
```

```
# New Data

print(" New shape : ", data_riverP.shape)
data_riverP.head()
```

Affichage en fonction des jours

In [ ]:
```
data_riverP[['W','dA','S','Q']][:1500].plot(subplots=True, fontsize=12, figsize=(14,
plt.grid()
plt.show()
```

In [ ]:
```
data_riverP["Q"][:1500].plot()
# plt.axhline(data_river["Q"][0],linestyle = "--",color='red',label = "1st day value
plt.grid()
plt.legend()
plt.xlabel("Days")
plt.ylabel("Q (m3/s)")

plt.show()
```

Partie 1 : Apprentissage

Sélectionner une seule rivière !

1 - Paramètres

In [ ]:
```
scale           = 1        # Percentage of dataset to be used (1=all)
train_prop      = 0.8     # Percentage for train (the rest being for the test)
sequence_len    = 16
batch_size      = 16

features        = ['W','dA','S','Q']
features_len    = len(features)
```

2 - Préparation des échantillons

In [ ]:
```
# ---- Train / Test

train_len=int(train_prop*len(data_riverP))

dataset_train = data_riverP.loc[ :train_len-1, features ]
dataset_test  = data_riverP.loc[train_len:,    features ]

# ---- x_train / y_train

x_train = dataset_train.drop("Q",axis = 1)
y_train  = dataset_train['Q']

# ---- x_test / y_test

x_test = dataset_test.drop("Q",axis = 1)
print(type(x_test))
y_test  = dataset_test['Q']
print(type(y_test))

# ---- Normalize

mean = x_train.mean()
std  = x_train.std()
```

```
for i in x_train.columns:
    if std[i]!=0:

        x_train[i] = (x_train[i] - mean[i]) / std[i]
        x_test[i]  = (x_test[i]  - mean[i]) / std [i]


print('Dataset       : ',data_riverP.shape)
print('Train dataset : ',dataset_train.shape)
print('Test  dataset : ',dataset_test.shape)
```

In [ ]:
```
x_test.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
```

3 - Data Generator

In [ ]:
```
# ---- Train generator
train_generator = TimeseriesGenerator(x_train, y_train, length=sequence_len,  batch_
test_generator  = TimeseriesGenerator(x_test , y_test , length=sequence_len,  batch_



# ---- Echantillons
train_x , train_y = train_generator[0]
test_x  , test_y  = test_generator[0]

# ---- About

print(f'Nombre de train batchs disponibles : ', len(train_generator))
print('batch x shape : ',train_x.shape)
print('batch y shape : ', train_y.shape)
```

4 - Création du modèle

In [ ]:
```
model = keras.models.Sequential()
model.add( keras.layers.InputLayer(input_shape=(sequence_len, features_len-1)) )
model.add( keras.layers.LSTM(10, activation='relu') )
model.add( keras.layers.Dense(1) )
model.summary()
```

In [ ]:
```
# model_drop = keras.models.Sequential()
# model_drop.add( keras.layers.InputLayer(input_shape=(sequence_len, features_len-1)
# model_drop.add( keras.layers.LSTM(10, activation='relu') )
# model_drop.add( keras.layers.Dropout(0.20))
# model_drop.add( keras.layers.Dense(1) )
# model_drop.summary()
```

Metrics

In [ ]:
```
def nRMSE(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1))/K.mean(y_true)

def NSE(y_true, y_pred):
    return 1- K.mean(K.square(y_pred-y_true))/K.mean(K.square(y_true-K.mean(y_true))

def R2(y_true,y_pred):
    return tfp.stats.correlation(y_true, y_pred)

def KGE(y_true,y_pred):
    beta = K.mean(y_pred)/K.mean(y_true)
```

```
    alpha = K.var(y_pred)/K.var(y_true)
    r2 = R2(y_true,y_pred)
    kge = 1 - K.sqrt(K.square(beta-1) + K.square(alpha-1)+K.square(r2-1))
    return kge
```

5 - Compilation

In [ ]:
```
model.compile(optimizer='adam',
              loss='mse',
              metrics   = ['mae','mse',nRMSE] )
```

In [ ]:
```
# model_drop.compile(optimizer='adam',
#                loss='mse',
#                metrics   = ['mae'] )
```

6 - Fit

In [ ]:
```
epochs = 10

history=model.fit(train_generator,
                  epochs=epochs,
                  verbose=1,
                  validation_data = test_generator)
```

In [ ]:
```
# epochs = 10

# history_drop=model_drop.fit(train_generator,
#                    epochs=epochs,
#                    verbose=1,
#                    validation_data = test_generator)
```

7 - Plot Metrics

In [ ]:
```
hist =pd.DataFrame(data=history.history)
# hist_drop =pd.DataFrame(data=history_drop.history)
```

In [ ]:
```
plt.figure(0)
plt.plot(hist['mae'],'-or')
plt.axhline(0.20*np.mean(y_train),linestyle = '--', color = 'green' , label = "20% e
plt.title("MAE")
plt.legend()
plt.ylabel("mae (m3/s)")
plt.grid()
plt.xlabel("Epochs")
plt.show()

plt.figure(1)
plt.plot(hist['mse'],'-or')
plt.title("MSE")
plt.xlabel("Epochs")
plt.grid()
plt.show()

plt.figure(2)
plt.plot(hist['nRMSE'],'-or')
plt.title("nRMSE")
plt.grid()
plt.xlabel("Epochs")
plt.show()
```

In [ ]:
```
# plt.figure(3)
```

```
# plt.plot(hist['R2'],'-or')
# plt.title("R2 sans dropout")
# plt.legend()
# plt.show()

# plt.figure(2)
# plt.plot(hist['KGE'],'-or')
# plt.title("KGE sans dropout")
# plt.legend()
# plt.show()

# plt.figure(2)
# plt.plot(hist['NSE'],'-or')
# plt.title("NSE sans dropout")
# plt.legend()
# plt.show()
```

In [ ]:
```
# plt.figure(1)
# plt.plot(hist_drop['mae'],'-or')
# plt.axhline(0.20*np.mean(y_train),linestyle = '--')
# plt.title("MAE avec dropout")
# plt.show()
```

8 - Prediction on n-days

In [ ]:
```
def nRMSE_P(y_true, y_pred):
    return np.sqrt(np.mean(np.square(y_pred - y_true)))/np.mean(y_true)
```

In [ ]:
```
def get_prediction(xtest, ytest ,  model, n_days= 5, plot = False):

    # ---- Initial sequence
    #
    s = sequence_len
    y_pred     = ytest[s:s+sequence_len].copy()
    y_true     = ytest[s:s+sequence_len+n_days].copy()

    # ---- Iterate
    #
    y_pred = list(y_pred)

    for i in range(n_days):
        x_sequence = xtest[s+i:s+sequence_len+i].copy()
        aux_pred = model.predict( np.array([x_sequence]))
        y_pred.append(aux_pred[0])

    y_true.reset_index(drop=True,inplace=True)
    if plot :
        plt.figure(figsize=(10,5))
        plt.plot(y_true,label = 'Target')
        plt.plot(y_pred,label = 'Prediction')
        plt.title("Prediction on " + str(n_days) + "  for " + river_name)
        plt.grid()
        plt.legend()
        plt.show()
        print("nRMSE : " , nRMSE_P(y_true, y_pred)[0])
    return y_true, y_pred
```

In [ ]:
```
# Sur train
true , pred = get_prediction(x_test,y_test,model,340, True)
```

In [ ]:
```
def Low_froude(river,pred):
    # Pred => si True calcule du Low-froude sur Qpred sinon sur Qtrue
```

```
    A0r = river["A0"]
    dArp = river["dA"]
    if pred:
        print("oui")
        Qrp = river["Qp"]
    else:
        Qrp = river["Q"]
    Srp = river["S"]
    Wrp = river["W"]
    Zrp = river["height"]
    Zr0 = np.min(Zrp)
    Wr0 = np.min(Wrp)
    c1rp = Wrp**(-2./5.) * Srp**(3./10.)
    c2rp = c1rp * dArp
    c3rp = (Zrp - Zr0)
    c4r = 1.0 / Wr0

    alpha = river["alpha"]
    beta = river["beta"]

    Qest = (alpha**(3./5.) * (c1rp * A0r + c2rp) * (c4r * A0r + c3rp)**(3./5. * beta

    return Qest
```

In [ ]:

```
Qtrue, Qpredict =get_prediction(x_test,y_test,model,data_river["Q"].shape[0])
data_river["Qp"] = Qpredict[:-sequence_len]
Q_LF = Low_froude(data_river,True)
```

In [ ]:

```
x = np.arange(0,data_river["Q"].shape[0]-sequence_len)
plt.figure(figsize = (10,6))
plt.scatter(x,Qpredict[:data_river["Q"].shape[0]-sequence_len],label="Prediction", c
plt.scatter(x,Q_LF[sequence_len:],label = "Low Froude", color ='g', s=3)
plt.plot(x,Qtrue[:data_river["Q"].shape[0]-sequence_len],label = "Target", color='b'
plt.legend()
plt.grid()
plt.show()
```

In [ ]:

```
import numpy.linalg as npl
```

In [ ]:

```
npl.norm(Qtrue[:data_river["Q"].shape[0]-sequence_len] - Q_LF[sequence_len:])
print(np.shape(Qtrue[:data_river["Q"].shape[0]-sequence_len].to_list()))
print(type(Q_LF[sequence_len:].to_numpy()))
print((np.subtract(Qtrue[:data_river["Q"].shape[0]-sequence_len].to_numpy(),Q_LF[seq
print(npl.norm(np.subtract(Q_LF[sequence_len:],Qpredict[:data_river["Q"].shape[0]-se
```

In [ ]:

```
nRMSE_P(Qtrue[:data_river["Q"].shape[0]-sequence_len], Q_LF[sequence_len:])
```

In [ ]:

```
nRMSE_P(Qpredict[:data_river["Q"].shape[0]-sequence_len],Qtrue[:data_river["Q"].shap
```

In [ ]:

```
nRMSE_P(Qpredict[:data_river["Q"].shape[0]-sequence_len], Q_LF[sequence_len:])
```