**Annexes :**

**code génération réseau :**

import tensorflow as tf

import pylab as plt

from math import cos,sin,sqrt

from time import time

from tensorflow.keras.callbacks import TensorBoard

import numpy as np

from datetime import datetime

#calcul moyenne ecart type

def variations(data):

R=[[0,0] for i in range(len(data[0]))]

for x in data:

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][0]+=x[i]

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][0]=R[i][0]/len(data)

for x in data:

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][1]+=(x[i]-R[i][0])\*\*2

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][1]=sqrt(R[i][1]/len(data))

return R

#mesure temps d'execution

debut=time()

data=np.load('data\_sim\_1.npy')

#melange des donnés pour eviter l'overfitting

np.random.shuffle(data)

#print(data[0])

#Normalization des données

R=variations(data)

for i in range(len(data)):

for j in range(len(data[0])-1):#issue constant input bigger dataset needed

data[i][j]=(data[i][j]-R[j][0])/R[j][1]

print(variations(data))

#separation training set validation set

proportion=int(len(data)\*0.9)

train=data[:proportion]

test=data[proportion:]

x\_train=train[:,2:5]

y\_train=train[:,:2]

x\_test=test[:,2:5]

y\_test=test[:,:2]

#architecture du modele

model =tf.keras.models.Sequential() # random architecture

#print(1)

#model.add(keras.layers.Flatten(input\_shape=(20,))) #useless after all

model.add(tf.keras.layers.Dense(3))

model.add(tf.keras.layers.Dense(64, activation='linear'))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(128, activation='relu'))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(64, activation='relu'))

model.add(tf.keras.layers.Dense(32, activation='relu'))

model.add(tf.keras.layers.Dense(16, activation='relu'))

model.add(tf.keras.layers.Dense(8 , activation='relu'))

model.add(tf.keras.layers.Dense(2 , activation='relu'))

#parametres d'apprentissage

opt = tf.keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-7, amsgrad=False)

# Compile model

model.compile(

loss='mean\_squared\_error',

optimizer=opt,

metrics=['accuracy'],

)

#memoire loss

print\_loss=[]

#création clef primaire pour nouveaux modeles

now = datetime.now() # current date and time

date\_time = now.strftime("%m/%d/%Y-%H:%M:%S")

for i in range(21):

#mesure temps d'execution

a=time()

#opt = tf.keras.optimizers.Adam(lr=1/(10000\*(i+1)-5000), beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-7, amsgrad=False)# sert a rien

#entrainement du réseau

history=model.fit(x\_train,

y\_train,

epochs=1000 ,

validation\_data=(x\_test, y\_test),

verbose=0,

batch\_size=900)

print\_loss.append(history.history['loss'])

print(date\_time,": ",time()-a," ",history.history['loss'][-1]," ",i)

#enregistrement etat du modèle

model.save\_weights(f"models\\my\_model\_weights\_var\_cor\_lower\_dim{date\_time.replace(':','\_').replace('/','\_')}.h5")

#prediction du modele

features = data[:,2:5]

predictions = model.predict(features)

np.save("predictions\_sim\_1.npy",predictions)

#affichage

Y=[]

X=[]

for x in data[:,:2]:

X.append((x[0]\*R[0][1]+R[0][0])\*cos((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y.append((x[0]\*R[0][1]+R[0][0])\*sin((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y1=[]

X1=[]

for x in predictions:

X1.append((x[0]\*R[0][1]+R[0][0])\*cos((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y1.append((x[0]\*R[0][1]+R[0][0])\*sin((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

plt.figure()

plt.plot(X,Y,'ro',X1,Y1,'bo')

finloss=[]

for x in print\_loss:

for y in x:

finloss.append(y)

plt.figure()

plt.plot(finloss,'b')

print(time()-debut)

plt.show()

j

**reprise entraînement :**

import tensorflow as tf

import pylab as plt

from math import cos,sin,sqrt

##test tensorboard

from time import time

from tensorflow.keras.callbacks import TensorBoard

import numpy as np

from datetime import datetime

def moyenneGlissante(L):

R=[]

halfsize=100

for i in range(halfsize,len(L)-halfsize):

R.append(sum(L[i-halfsize:i+halfsize])/halfsize/2)

return R

def variations(data):

R=[[0,0] for i in range(len(data[0]))]

for x in data:

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][0]+=x[i]

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][0]=R[i][0]/len(data)

for x in data:

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][1]+=(x[i]-R[i][0])\*\*2

for i in range(len(data[0])-1):#issue constant input bigger dataset needed

R[i][1]=sqrt(R[i][1]/len(data))

return R

debut=time()

data=np.load('data\_sim\_1.npy')

np.random.shuffle(data)

#print(data[0])

R=variations(data)

for i in range(len(data)):

for j in range(len(data[0])-1):

data[i][j]=(data[i][j]-R[j][0])/R[j][1]

print(variations(data))

proportion=int(len(data))

train=data[:proportion]

test=data[proportion:]

x\_train=train[:,2:5]

y\_train=train[:,:2]

x\_test=test[:,2:5]

y\_test=test[:,:2]

model =tf.keras.models.Sequential() # random architecture

#print(1)

#model.add(keras.layers.Flatten(input\_shape=(20,))) #useless after all

model.add(tf.keras.layers.Dense(3,input\_shape=(3,)))

model.add(tf.keras.layers.Dense(64, activation='linear'))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(1024, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

#print(2)

model.add(tf.keras.layers.Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(64, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(32, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(16, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(8 , activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(2 , activation='linear'))

model\_name="models\\my\_model\_weights\_var\_cor05\_04\_2019-18\_03\_55.h5"

model.load\_weights(model\_name)

opt = tf.keras.optimizers.Adam(lr=0.0000005, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0, amsgrad=False)

# Compile model

model.compile(

loss='mean\_squared\_error',

optimizer=opt,

metrics=['accuracy'],

)

print\_loss=[]

for i in range(64):

a=time()

#opt = tf.keras.optimizers.Adam(lr=0.000000000001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-7, amsgrad=False)#doesn't do anything

history=model.fit(x\_train,

y\_train,

epochs=10 ,

validation\_data=(x\_test, y\_test),

verbose=1,

batch\_size=1)

now = datetime.now() # current date and time

date\_time = now.strftime("%m/%d/%Y-%H:%M:%S")

print\_loss.append(history.history['loss'])

model.save\_weights(model\_name)

np.save("finloss.npy",print\_loss)

print(date\_time,": ",int(time()-a)," ",sum(history.history['loss'])/len(history.history['loss'])," ",i)

#model.save\_weights(f"models\\my\_model\_weights\_var\_cor05\_04\_2019-18\_03\_55.h5")

features = data[:,2:5]

predictions = model.predict(features)

np.save("predictions\_sim\_1.npy",predictions)

Y=[]

X=[]

for x in data[:,:2]:

X.append((x[0]\*R[0][1]+R[0][0])\*cos((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y.append((x[0]\*R[0][1]+R[0][0])\*sin((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y1=[]

X1=[]

for x in predictions:

X1.append((x[0]\*R[0][1]+R[0][0])\*cos((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

Y1.append((x[0]\*R[0][1]+R[0][0])\*sin((x[1]\*R[1][1]+R[1][0])/360\*2\*3.14))

plt.figure()

plt.plot(X,Y,'ro',X1,Y1,'bo')

finloss=[]

for x in print\_loss:

for y in x:

finloss.append(y)

plt.figure()

plt.plot(moyenneGlissante(finloss),'b')

plt.figure()

plt.plot(finloss,'b')

print(time()-debut)

plt.show()

**réseau modèle analytique (marche a quelques détails près):**

from random import \*

import numpy as np

import tensorflow as tf

import pylab as plt

from math import cos,sin

from time import time

from datetime import datetime

#from tensorflow import keras

###

def calculSortie(L):

S , i = L[0],L[1]

a=[i[2],0]

b=[i[2]\*i[1],i[0]]

return [

(a[0]\*S[0][0]+b[0]\*S[3][0])\*\*2+(a[1]\*S[0][1]+b[1]\*S[3][1])\*\*2,

(a[0]\*S[1][0]+b[0]\*S[5][0])\*\*2+(a[1]\*S[1][1]+b[1]\*S[5][1])\*\*2,

(a[0]\*S[2][0]+b[0]\*S[6][0])\*\*2+(a[1]\*S[2][1]+b[1]\*S[6][1])\*\*2,

(a[0]\*S[3][0]+b[0]\*S[7][0])\*\*2+(a[1]\*S[3][1]+b[1]\*S[7][1])\*\*2

]

# re im

#Styp=np.array([1,0,0.7,1,]) entrer typical S param

Num=[1]

"""ds l'ordre S13,S14,S15,S16,S23,S24,S25,S26 """

def generSparam(rang,x): #create Sparam "around" the typical ones

Styp=[[-0.5,0.0],[0.0,0.5],[-0.5,0.0],[0.0,0.5],[0.5,0.0],[0.0,0.5],[0.0,0.5],[-0.5,0.0]]

num = Num[0]

Num[0]+=1

S=[]

for j in range(8):

S.append([Styp[j][0]+rang/x\*num-rang/2,Styp[j][1]+rang/x\*num-rang/2])

"""S=Styp+range/num\*i+range/num\*i\*1j"""

return S.copy()

def flatten(A):

L=[]

for i in A:

for j in i:

L.append(j)

return L

def features2Array(x):

X=np.zeros((20))

for i in range(20):

X[i]=x[i]

return X

def labels2Array(x):

X=np.zeros((4))

for i in range(4):

X[i]=x[i]

return X

def generInput():#create random but likely inputs

return [random()\*20-10,random()\*20-10,random()\*20-10] # rep phase, rap amplitude, puissance d'entree en racine

def generS():# create random S param

return [[random()\*2-1,random()\*2-1] for i in range(8)]

def generDataset(len):

Dataset=[]

for x in range(len):

S=generSparam(len,x+1)

i=generInput()

x=np.array(flatten(S)+calculSortie([S,i]))

y=np.array(i)

Dataset.append([x,y])

return Dataset

def split(Dataset):

X,Y=[],[]

for i in Dataset:

X.append(i[0])

Y.append(i[1])

return X,Y

trainlen=200000

train=generDataset(trainlen)

x\_train, y\_train = split(train) ## .copy() ??

x\_train, y\_train=np.array(x\_train), np.array(y\_train)

now = datetime.now() # current date and time

date\_time = now.strftime("%m/%d/%Y-%H:%M:%S")

#np.save(f"x-train\_len-{trainlen}\_date-{date\_time.replace(':','\_').replace('/','\_')}",x\_train)

#np.save(f"y-train\_len-{trainlen}\_date-{date\_time.replace(':','\_').replace('/','\_')}",y\_train)

test=generDataset(200)

x\_test, y\_test = split(test)

x\_test, y\_test=np.array(x\_test), np.array(y\_test)

print(y\_test.shape)

print(x\_test.shape)

# model

a=time()

model =tf.keras.models.Sequential() # random architecture

#print(1)

#model.add(keras.layers.Flatten(input\_shape=(20,))) #useless after all

model.add(tf.keras.layers.Dense(20))

model.add(tf.keras.layers.Dense(64, activation='linear'))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(1024, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

#print(2)

model.add(tf.keras.layers.Dense(512, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

#model.add(tf.keras.layers.Dropout(0.01)) # prevent overfitting lol value might be a little bit high

model.add(tf.keras.layers.Dense(128, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(64, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(32, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(16, activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(8 , activation=tf.keras.layers.LeakyReLU(alpha=0.3)))

model.add(tf.keras.layers.Dense(3 , activation='linear'))

opt = tf.keras.optimizers.Adam(lr=0.001, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-7, amsgrad=False)

# Compile model

model.compile(

loss='mean\_squared\_error',

optimizer=opt,

metrics=['accuracy'],

)

print\_loss=[]

for i in range(1):

now = datetime.now() # current date and time

date\_time = now.strftime("%m/%d/%Y-%H:%M:%S")

opt = tf.keras.optimizers.Adam(lr=1/100/10\*\*i, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=1e-7, amsgrad=False)

history=model.fit(x\_train,

y\_train,

epochs=1000 ,

validation\_data=(x\_test, y\_test),

verbose=1,

batch\_size=20000)

#model.save\_weights(f"models\\my\_model\_weights{date\_time.replace(':','\_').replace('/','\_')}.h5")

print\_loss.append(history.history['loss'])

features = x\_test

predictions = model.predict(features)

np.save("predictions\_sim\_1.npy",predictions)

Y=[]

X=[]

for x in x\_test:

X.append((x[0]\*1.1+0.55)\*cos((x[1]+0.5)\*2\*3.14))

Y.append((x[0]\*1.1+0.55)\*sin((x[1]+0.5)\*2\*3.14))

Y1=[]

X1=[]

for x in predictions:

X1.append((x[0]\*1.1+0.55)\*cos((x[1]+0.5)\*2\*3.14))

Y1.append((x[0]\*1.1+0.55)\*sin((x[1]+0.5)\*2\*3.14))

plt.figure()

plt.plot(X,Y,'ro',X1,Y1,'bo')

finloss=[]

for x in print\_loss:

for y in x:

finloss.append(y)

plt.figure()

plt.plot(finloss,'b')

b=time()

print(time()-a)

plt.show()

**Mise en forme des données :**

import numpy as np

f=open("input.csv", "r")

lines =f.readlines()

"""

for x in lines:

print(x)

"""

#print(len(lines[1:]))

#print(lines[46:])

content=lines[46:]

data=np.zeros((len(content),6))

for i in range(len(content)):

line=content[i]

liste=line.split(',')

#print(liste)

data[i][0]=float(liste[0])

data[i][1]=float(liste[1])

data[i][2]=float(liste[3])

data[i][3]=float(liste[4])

data[i][4]=float(liste[5])

data[i][5]=float(liste[6])

#print(data[0])

np.save("data\_sim\_1.npy",data)

**Calibration par correction de perspective :**

import numpy as np

import cv2

import pylab as plt

from math import cos,sin,sqrt

def calculXY(L):

point=[]

for i in L:

point.append([ (i[4]-i[3])/sqrt(2)/i[5] , (i[3]+i[4]-i[2]-i[5])/2/sqrt(2)/i[5] ])

return point

data=np.load('data\_sim\_1.npy')

#data=data[:500]

Y=[]

X=[]

for x in data[:,:2]:

X.append((x[0])\*cos((x[1])/360\*2\*3.14))

Y.append((x[0])\*sin((x[1])/360\*2\*3.14))

plt.figure()

plt.plot(X,Y,'ro')

print(data[400])

point=calculXY(data)

X1=[i[0] for i in point]

Y1=[i[1] for i in point]

print(len(X))

plt.plot(X1,Y1,'go')

#plt.show()

pointPicked=[(len(data)\*(i+1))//100 for i in range(99)]

pts\_src=np.array([point[i] for i in pointPicked])

pts\_dst=np.array([[X[i],Y[i]]for i in pointPicked])

h, status = cv2.findHomography(pts\_src, pts\_dst)

for i in range(len(point)):

point[i].append(1)

#print(point)

pointCori=[]

print(h,type(h))

for i in range(len(point)):

pointCori.append(np.dot(h,np.array(point[i])))

print(pointCori)

XCor=[pointCori[i][0]/pointCori[i][2] for i in range(len(pointCori))]

YCor=[pointCori[i][1]/pointCori[i][2] for i in range(len(pointCori))]

plt.plot(XCor,YCor,'bo')

plt.show()