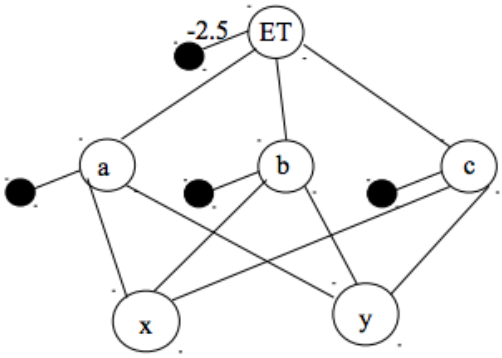
Projet Apprentissage Automatique

Objectif : Réalisation en python d'un réseau de neurone capable de reconnaitre une zone convexe



Réalisation :

```
import numpy as np

X = np.array(
    [[1, 1], [1, 2], [1, 3], [1, 4], [1, 5],
      [2, 1], [2, 2], [2, 3], [2, 4], [2, 5],
      [3, 1], [3, 2], [3, 3], [3, 4], [3, 5],
      [4, 1], [4, 2], [4, 3] ,[4, 4] ,[4, 5],
     ])

y = np.array([
      [0], [0], [0], [0], [0],
      [0], [0], [1], [0], [0],
      [0], [1], [1], [0], [0],
      [0], [0], [0], [0], [0],
     ])


#Sigmoid Function
def sigmoid (x):
    return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

# Parametres
epoch=800000
lr=1
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))

wout = np.array([[1.], [1.], [1.]])
bout = np.array([-2.5])

for i in range(epoch):
    #Forward Propogation
    hidden_layer_input=np.dot(X,wh) + bh
    hiddenlayer_activations = sigmoid(hidden_layer_input)
    output_layer_input= np.dot(hiddenlayer_activations,wout)+ bout
    output = sigmoid(output_layer_input)

    #Backpropagation
    E = y-output
    d_output = E * derivatives_sigmoid(output)
    d_hiddenlayer = d_output.dot(wout.T) * derivatives_sigmoid(hiddenlayer_activations)

    wout += hiddenlayer_activations.T.dot(d_output) *lr
    bout += np.sum(d_output, axis=0) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0) *lr
```

Test

```
for x in X :
    hidden_layer_input=np.dot(x,wh) + bh
    hiddenlayer_activations = sigmoid(hidden_layer_input)
    output_layer_input= np.dot(hiddenlayer_activations,wout)+ bout
    output = sigmoid(output_layer_input)
    print(x, " -> ", output)
```

```
[1 1]  ->  [[ 0.00043967]]
[1 2]  ->  [[ 0.0004542]]
[1 3]  ->  [[ 0.00048376]]
[1 4]  ->  [[ 0.0009405]]
[1 5]  ->  [[ 0.00010521]]
[2 1]  ->  [[  1.48103862e-08]]
[2 2]  ->  [[ 0.00209404]]
[2 3]  ->  [[ 0.99784481]]
[2 4]  ->  [[ 0.00120226]]
[2 5]  ->  [[ 0.00020471]]
[3 1]  ->  [[  2.28955130e-05]]
[3 2]  ->  [[ 0.99905867]]
[3 3]  ->  [[ 0.9970925]]
[3 4]  ->  [[ 0.00021559]]
[3 5]  ->  [[ 0.0002046]]
[4 1]  ->  [[ 0.00144191]]
[4 2]  ->  [[ 0.00165222]]
[4 3]  ->  [[ 0.0022327]]
[4 4]  ->  [[ 0.00020487]]
[4 5]  ->  [[ 0.00020459]]
```

Affichage

```
from matplotlib import pyplot as plt

class1 = [[1, 1], [1, 2], [1, 3], [1, 4], [1, 5],
      [2, 1], [2, 2], [2, 4], [2, 5],
      [3, 1], [3, 4], [3, 5],
      [4, 1], [4, 2], [4, 3] ,[4, 4] ,[4, 5],
     ]
class2 = [(2, 3), (3, 2), (3, 3)]


X, Y = zip(*class1)
plt.scatter(X, Y, c="r")

X, Y = zip(*class2)
plt.scatter(X, Y, c="b")

x = [0, 5]
y = [(wh[0][0]/-wh[1][0])*0  + (bh[0][0]/-wh[1][0]), (wh[0][0]/-wh[1][0])*5  + (bh[0][0]/-wh[1][0])]
plt.plot( x, y, color="black" )

x = [0, 5]
y = [(wh[0][2]/-wh[1][2])*0  + (bh[0][2]/-wh[1][2]), (wh[0][2]/-wh[1][2])*5  + (bh[0][2]/-wh[1][2])]
plt.plot( x, y, color="black" )


x = [0, 5]
y = [(wh[0][1]/-wh[1][1])*0  + (bh[0][1]/-wh[1][1]), (wh[0][1]/-wh[1][1])*5  + (bh[0][1]/-wh[1][1])]
plt.plot( x, y, color="black" )

plt.show()
```