Homework 6: Perceptron due on 04/11/2023

Please study the note on the sample python code and modify it to answer the question. Please print out your result and submit it in class.

TA email for emergency: 蕭方凱 zeus950068@gmail.com

HW6 problem

1. (40%) Implementation of an inverter to do NOT operation

In the lecture, I have given an example on how to train a perceptron to perform OR operation. Please modify the code to train a perceptron to perform AND operation.

Use the following parameters:

Setting HIGH and LOW to be 0.9 and 0.1. Therefore input X and desired output NOT is given by

X	NOT X
0.1	0.9
0.9	0.1

When setting the input to the perceptron, please add Gaussian noise to X1 and X2 with standard deviation 0.1.

We use a perceptron with sigmodal activation function defined by

$$sigmod(\eta) = \frac{1}{1 + \exp(-\eta)}$$

The output of the perceptron y is given by

$$y = sigmod(wx + w_0)$$

First, initialize weight coefficient from uniform distribution U(-0.1,0.1)

and use weight update rule to find weight coefficient w and w0. Learning rate = 0.1 and 1000 iteration for each case, ie., X= 0.1 or X=0.9

When training is finished, print out the weight coefficients and

print out output y for two cased and y is calculated by

$y = sigmod(wx + w_0)$ x=0.1

$y = sigmod(wx + w_0)$ x=0.9

Solution:

wx -4.447 w0 2.187

x 0.1 y 0.851

x 0.9 y 0.14

```
import random
from math import *
```

```
eta=0.1 #define learning rate
# intialization of weight coeffient
w_x=random.uniform(-0.1,0.1)
theta=random.uniform(-0.1,0.1)

#define HIGH and LOW
HIGH= 0.9
LOW = 0.1

def logistic(w_x, theta, x):
        return 1/(1+exp(-w_x*x-theta))

#feed in training sample type 1
#calcuated expected output

for i in range(1000):
        x00=random.gauss(0,0.1)
        output_cal=logistic(w_x,theta,x00)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x00
        theta= theta-eta*(output_cal-HIGH)*1

        x11=random.gauss(1,0.1)

        output_cal=logistic(w_x,theta, x11)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x11     #cheng output level to 0.9
        theta= theta-eta*(output_cal-LOW)*1
# the following code is used for debugging
        if i % 100==0:
                # error calculation sample at two points x=0 and x=1
                error=0
                x_test=0.1
                error= error+(logistic(w_x,theta, x_test)-HIGH)**2
```

```
        x_test=0.9
        error= error+(logistic(w_x,theta, x_test)-LOW)**2
        print('iteration',i, 'error', error)
# END OF ERROR CALCULATION CODE

print('wx', round(w_x,3), 'theta', round(theta,3))
# error calculation
```

2. (60%) Implement the training of a perceptron to perform AND operation

In the lecture, I have given an example on how to train a perceptron to perform OR operation. Please modify the code to train a perceptron to perform AND operation. Few modification is needed.This time please set HIGH to be 0.9 and LOW to be 0.1 as the desired output.In other words, HIGH corresponds 1 in Boolean operation and LOW corresponds to 0 in Boolean operation.

(In our OR example, we set the definition of HIGH to be 0.99 and LOW to be 0.1.)

For AND operation, we know the truth table for X1 AND X2 is given

By

X1	X2	X1 AND X2
0	0	0
1	0	0
0	1	0
1	1	1

Assuming X1 and X2 are Boolean. Here in perceptron (or in real-world digital electronics), we represent X1 and X2 as in

X1	X2	X1 AND X2
0.1	0.1	0.1
0.9	0.1	0.1
0.1	0.9	0.1
0.9	0.9	0.9

When setting the input to the perceptron, please add Gaussian noise to X1 and X2 with standard deviation 0.1.

We use a perceptron with sigmodal activation function

$$sigmod(x) = \frac{1}{1 + \exp{(-x)}}$$

The output of the perceptron y is given by

$$y = sigmod(w_1 x_1 + w_2 x_2 + w_0)$$

Intialize w1, w2, and w0 from a uniform distribution from -0.1 and 0.1. (The same as our lecture example.)Use 1000 iteration for each pair of inputs , there will be 4000 iteration. (The same as our lecture OR example.) There are four input (0.1, 0.1), (0.9,0.1), (0.1, 0.9), (0.9, 0.9).

Print out the weight coefficients after training and plot the decision boundary from weight coefficient.

Solution:

```
wx 2.772 wy 2.791 theta -4.141
x 0 y 0 output 0.015650253807040257
x 1 y 0 output 0.20276053074147007
x 0 y 1 output 0.2057172582850203
x 1 y 1 output 0.8055621629042656
```

decision boundary is given by $w_1 x_1 + w_2 x_2 + w_0 = 0$

you will get something like x1+x2- 1.5 =0

```
import random
from math import *
print('exp(0)', round(exp(1),4)) #verify exponential function is working

eta=0.1 #define learning rate
# intialization of weight coeffient
w_x=random.uniform(-0.1,0.1)
w_y=random.uniform(0.1,0.1)
theta=random.uniform(-0.1,0.1)

#define HIGH and LOW
HIGH= 0.9
LOW = 0.1

def logistic(w_x,w_y, theta, x,y):
    return 1/(1+exp(-w_x*x-w_y*y-theta))
```

```
#feed in training sample type 1
#calcuated expected output

for i in range(1000):
        x00=random.gauss(0,0.1)
        y00=random.gauss(0,0.1)
        output_cal=logistic(w_x,w_y,theta, x00,y00)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x00
        w_y=w_x-eta*(output_cal-LOW)*y00
        theta= theta-eta*(output_cal-LOW)*1

        x10=random.gauss(1,0.1)
        y10=random.gauss(0,0.1)

        output_cal=logistic(w_x,w_y,theta, x10,y10)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x10        #change here for other logic gate
        w_y=w_x-eta*(output_cal-LOW)*y10
        theta= theta-eta*(output_cal-LOW)*1

        x01=random.gauss(0,0.1)
        y01=random.gauss(1,0.1)

        output_cal=logistic(w_x,w_y,theta, x01,y01)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x01 #change here for other logic gate
        w_y=w_x-eta*(output_cal-LOW)*y01
        theta= theta-eta*(output_cal-LOW)*1

        x11=random.gauss(1,0.1)
        y11=random.gauss(1,0.1)

        output_cal=logistic(w_x,w_y,theta, x11,y11)
#delta rule
```

```
        w_x=w_x-eta*(output_cal-HIGH)*x11
        w_y=w_x-eta*(output_cal-HIGH)*y11
        theta= theta-eta*(output_cal-HIGH)*1
        #samping the error every 100 sample
        if i % 100==0:      #index divided by 10 = 0
            # error calculation
            error=0
            x_test=0
            y_test=0
            error= error+(logistic(w_x, w_y,theta, x_test, y_test)-LOW)**2    #for AND
expected outcome LOW

            x_test=1
            y_test=0
            error= error+(logistic(w_x, w_y,theta, x_test, y_test)-LOW)**2 #for AND
expected outcome LOW

            x_test=0
            y_test=1
            error= error+(logistic(w_x, w_y,theta, x_test, y_test)-LOW)**2 #for AND
expected outcome LOW

            x_test=1
            y_test=1
            error= error+(logistic(w_x, w_y,theta, x_test, y_test)-HIGH)**2 #for AND
expected outcome HIGH
            print('iteration',i, 'error', error)

print('wx', round(w_x,3), 'wy', round(w_y,3), 'theta', round(theta,3))
# calculate expect output
x_test=0
y_test=0
test_output= logistic(w_x, w_y,theta, x_test, y_test)
print('x', x_test, 'y', y_test, 'output', test_output)
```

```
x_test=1
y_test=0
test_output= logistic(w_x, w_y,theta, x_test, y_test)
print('x', x_test, 'y', y_test, 'output', test_output)

x_test=0
y_test=1
test_output= logistic(w_x, w_y,theta, x_test, y_test)
print('x', x_test, 'y', y_test, 'output', test_output)
x_test=1
y_test=1
test_output= logistic(w_x, w_y,theta, x_test, y_test)
print('x', x_test, 'y', y_test, 'output', test_output)
```