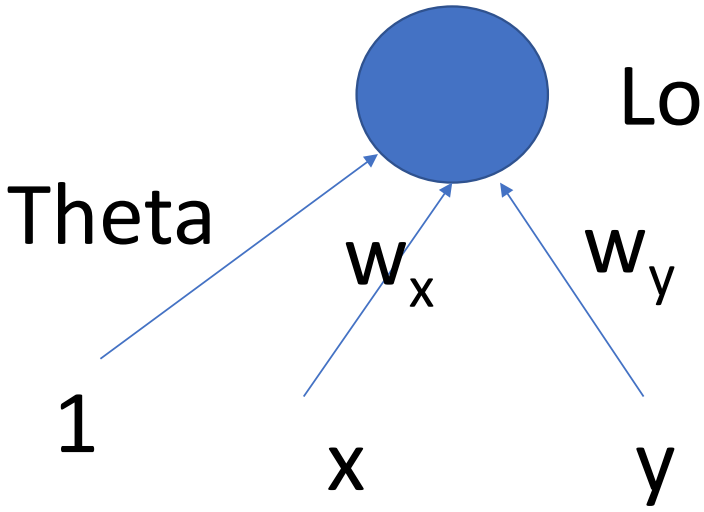
Perceptron OR gate

03-28-2023

YA TANG YANG

Single perceptron to implement OR



Training data
0 OR 0 = LOW
1 OR 0 = HIGH
0 OR 1 = HIGH
1 OR 1 = HIGH

Define

HIGH 0.99
LOW 0.01

Perceptron training

- Train a perceptron to do OR operation
- Logic OR gate with input polluted with Gaussian noise
- there are four type inputs
- # 0 OR 0 = LOW
- # 1 OR 0 = HIGH
- # 0 OR 1 = HIGH
- # 1 OR 1 = HIGH

Delta rule

- Generate a sample for input (0,0) and update
- Generate a sample for input (1,0) and update
- Generate a sample for input (0,1) and update
- Generate a sample for input (1,1) and update

Generate a sample for input (0,0)

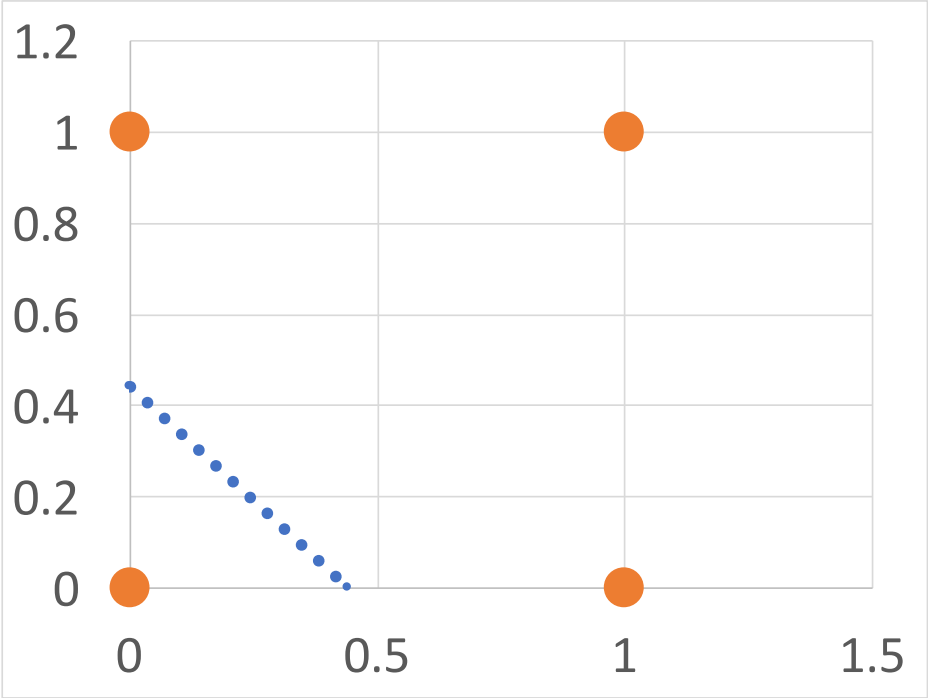
```
        x00=random.gauss(0,0.1)
        y00=random.gauss(0,0.1)
        output_cal=logistic(w_x,w_y,theta, x00,y00)
#delta rule
        w_x=w_x-eta*(output_cal-LOW)*x00
        w_y=w_x-eta*(output_cal-LOW)*y00
        theta= theta-eta*(output_cal-LOW)*1
```

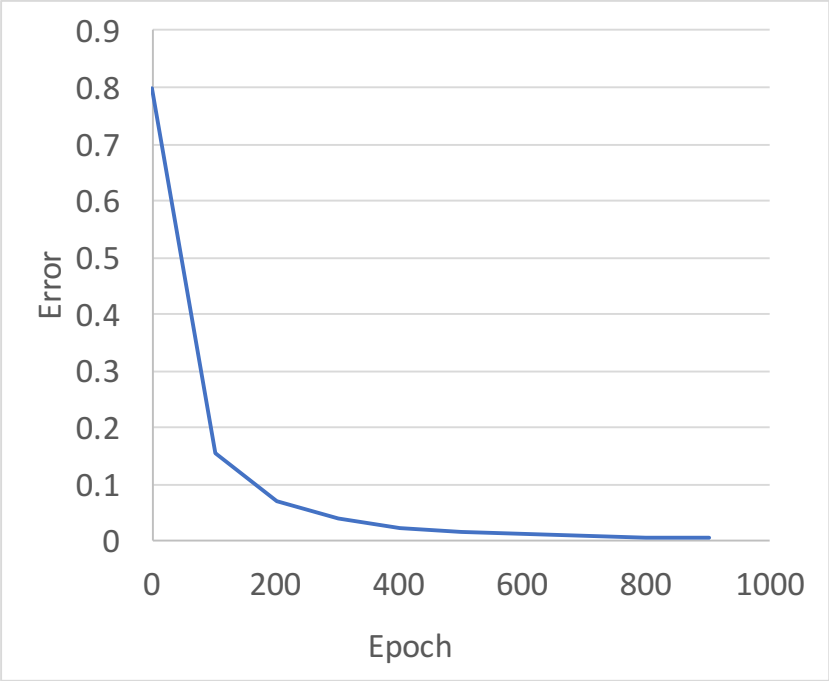
Delta rule:

Update = -Learning rate(Actual Output-Desired output)

Final weight coeffiecent

wx 6.09 wy 6.09 theta -2.693





Test the neuron for input

```
x 0 y 0 output 0.06336497914983781
x 1 y 0 output 0.9677346756607005
x 0 y 1 output 0.9677031963574627
x 1 y 1 output 0.9999247262916799
```

```
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= 0.99
LOW = 0.01

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-HIGH)*x10    #change here for other logic gate
    w_y=w_x-eta*(output_cal-HIGH)*y10
    theta= theta-eta*(output_cal-HIGH)*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-HIGH)*x01 #change here for other logic gate
    w_y=w_x-eta*(output_cal-HIGH)*y01
    theta= theta-eta*(output_cal-HIGH)*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 OR expected outcome LOW

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

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

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

```
print('wx', w_x, 'wy', w_y, 'theta', theta)
# 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)
```