```
import math
import random
import time

def main():
    #NEGATION
    print("NEGATION")
    cases = [(0,1), (1,0)]
    weights = [1] * 2
    neural(cases, weights)

    #OR
    print("OR")
    cases = [[0,0,0], [0,1,1], [1,0,1], [1,1,1]]
    weights = [1] * 3
    neural(cases, weights)

    #AND
    print("AND")
    cases = [[0,0,0], [0,1,0], [1,0,0], [1,1,1]]
    weights = [1] * 3
    neural(cases, weights)

    #XOR
    print("XOR")
    cases = [[0,0,0], [0,1,1], [1,0,1], [1,1,0]]
    weights = [1] * 3
    neural(cases, weights)

def neural(cases, weights):
    dw = 0.1
    flag = 0
    error = 0
    #0 = change w0 || 1 = change w1 || 2 = change w2
    for iterations in range(0, 10000):
        error = 0
        #measure error
        for case in cases:
            error = error + calcError(case, weights)
        error = error * 0.5

        #change weights to decrease error
        newError = 0
        tempWeights = weights[:]
        tempWeights[flag] = tempWeights[flag] + dw

        for case in cases:
            newError = newError + calcError(case, tempWeights)
        newError = newError * 0.5
        dE = (newError - error)/dw
```

```
        #adjust weights
        if dE > 0:
            weights[flag] = weights[flag] - dw
        elif dE < 0:
            weights[flag] = weights[flag] + dw
        flag = flag + 1
        if flag == len(weights):
            flag = 0
    print("error: ", error)
    print(weights, "\n")



def calcError(case, weights):
    toAdd = weights[0]
    for i in range(1, len(weights)):
        toAdd = toAdd + weights[i]*case[i-1]
    error = (case[1] - (1/(1 + ((math.e)**(-1 * toAdd)))))**2
    return error




main()
```