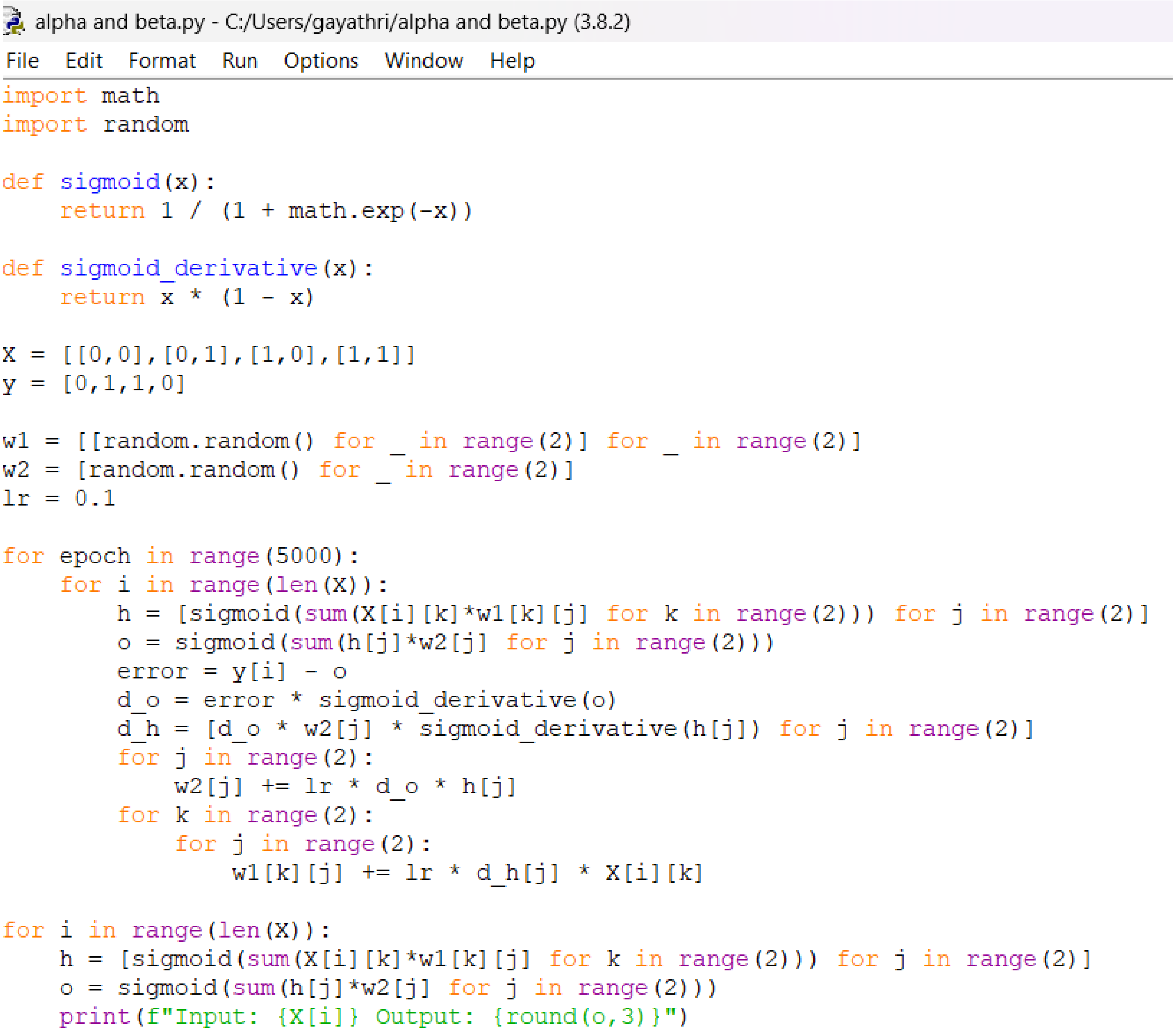
AIM

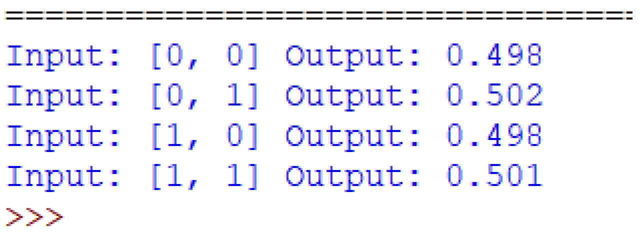
To implement a simple **Feedforward Neural Network** (2-2-1 architecture) in Python without using external libraries like NumPy and train it to solve the XOR problem.

ALGORITHM

1. **Initialize Data**: Input (X) = [[0,0],[0,1],[1,0],[1,1]], Output (y) = [0,1,1,0].
2. **Initialize Weights**: Random small weights for input→hidden (2x2) and hidden→output (2x1).
3. **Activation Function**: Use **Sigmoid** and its derivative.
4. **Training (Backpropagation)**:
    a. For each input:
        i. Compute hidden layer outputs.
        ii. Compute final output.
        iii. Calculate error = target – output.
        iv. Backpropagate error using gradient descent.
        v. Update weights accordingly.
5. **Repeat**: Train for multiple epochs to minimize error.
6. **Prediction**: After training, feed inputs again to check learned outputs.


alpha and beta.py - C:/Users/gayathri/alpha and beta.py (3.8.2)
File Edit Format Run Options Window Help

```
import math
import random

def sigmoid(x):
    return 1 / (1 + math.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

X = [[0,0],[0,1],[1,0],[1,1]]
y = [0,1,1,0]

w1 = [[random.random() for _ in range(2)] for _ in range(2)]
w2 = [random.random() for _ in range(2)]
lr = 0.1

for epoch in range(5000):
    for i in range(len(X)):
        h = [sigmoid(sum(X[i][k]*w1[k][j] for k in range(2))) for j in range(2)]
        o = sigmoid(sum(h[j]*w2[j] for j in range(2)))
        error = y[i] - o
        d_o = error * sigmoid_derivative(o)
        d_h = [d_o * w2[j] * sigmoid_derivative(h[j]) for j in range(2)]
        for j in range(2):
            w2[j] += lr * d_o * h[j]
        for k in range(2):
            for j in range(2):
                w1[k][j] += lr * d_h[j] * X[i][k]

for i in range(len(X)):
    h = [sigmoid(sum(X[i][k]*w1[k][j] for k in range(2))) for j in range(2)]
    o = sigmoid(sum(h[j]*w2[j] for j in range(2)))
    print(f"Input: {X[i]} Output: {round(o,3)}")
```



```
==================================
Input: [0, 0] Output: 0.498
Input: [0, 1] Output: 0.502
Input: [1, 0] Output: 0.498
Input: [1, 1] Output: 0.501
>>>
```

RESULT

After training for ~5000 epochs, the network learns the XOR function