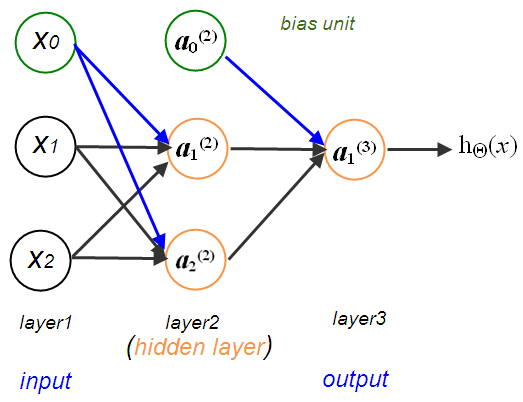
**Experiment No.: 04**

**Aim:** Write a program to implement logical XOR using the Perceptron network



**Steps:**

1.     **Initialization:**

·        Initialize weights (**w1** and **w2**) and bias (**bias**) to some values (provided as arguments to the function).

·        Calculate the linear combination of inputs and weights:

**linear\_combination = w1 \* x[0] + w2 \* x[1] + bias**.

·        Determine the output based on a threshold:

**output = 1 if linear\_combination >= threshold else 0**.

·        Set the target output as the logical OR of the inputs: **target\_output = x[0] xor x[1]**.

**Weight Update:**

If the output does not match the target output:

·        Update weights using the perceptron learning rule:

·        **w1 = w1 + learning\_rate \* (target\_output - output) \* x[0]**

·        **w2 = w2 + learning\_rate \* (target\_output - output) \* x[1]**

·        Clear the **outputs** list to start fresh for the next iteration.

·        Recursively call the **cal\_output\_and** function with updated weights to continue the learning process.

**Code:**

import numpy as np

import pandas as pd

def Xor(x, w11, w12, w21, w22, v1, v2, b1, b2, b3, count=1, max\_iterations=100, learning\_rate=0.5):

outputs = []

while count < max\_iterations:

target = -1 if (x[0] ^ x[1]) == 0 else 1

# Calculate zin1 and zin2

zin1 = x[0] \* w11 + x[1] \* w21 + b1

zin2 = x[0] \* w12 + x[1] \* w22 + b2

# Activation function: z=1 if >=0 else -1

activation = lambda z: -1 if z < 0 else 1

# Apply activation function to zin1 and zin2

z1 = activation(zin1)

z2 = activation(zin2)

# Calculate yin

yin = (z1 \* v1) + (z2 \* v2) + b3

y = activation(yin)

# Update weights if necessary

if y != target:

w11 += learning\_rate \* (target - y) \* x[0]

w21 += learning\_rate \* (target - y) \* x[1]

b1 += learning\_rate \* (target - y)

w12 += learning\_rate \* (target - y) \* x[0]

w22 += learning\_rate \* (target - y) \* x[1]

b2 += learning\_rate \* (target - y)

b3 += learning\_rate \* (target - y)

outputs.append([x[0], x[1], target, zin1, zin2, z1, z2, yin, y, w11, w21, b1, w12, w22, b2, b3])

# Check for convergence

if y == target:

print("Convered after", count, "iterations")

break

count += 1

output\_frame = pd.DataFrame(outputs, columns=['Input1', 'Input2', 'Target', 'zin1', 'zin2', 'Z1', 'Z2', 'Yin', 'Output', 'w11', 'w21', 'b1', 'w12', 'w22', 'b2', 'b3'])

print(output\_frame.to\_string(index=False))

w11 = 1

w12 = 1

w21 = 1

w22 = 1

v1 = 0.5

v2 = 0.5

b1 = 0.5

b2 = 0.3

b3 = 0.5

# Run the XOR function

test1 = np.array([1, 1])

test2 = np.array([1, -1])

test3 = np.array([-1, 1])

test4 = np.array([-1, -1])

print("\n",test1)

Xor(test1, w11, w12, w21, w22, v1, v2, b1, b2, b3)

print("\n",test2)

Xor(test2, w11, w12, w21, w22, v1, v2, b1, b2, b3)

print("\n",test3)

Xor(test3, w11, w12, w21, w22, v1, v2, b1, b2, b3)

print("\n",test4)

Xor(test4, w11, w12, w21, w22, v1, v2, b1, b2, b3)