**Problem Statement:**

Apply perceptron for realization of logic gates. (bias = 1)

**Procedure**:

Step 0: Initialize weights and biases to random number between 0 and 1 for all layers.

Step 1: Forward Pass (Propagation)

For each input neuron calculate the weighted sum for the hidden layer neuron :

Where Wi is the corresponding weight and

For output will be:

Step 2: Calculate the Error (Loss)

Compute the error between the predicted output​ and the actual target label​ using the Mean Squared Error (MSE):

Step 4: Update Weights and Biases

Update the weights and biases using the Gradient Descent rule:

Update weights for each layer:

Update biases for each layer:

Step 5: Repeat the Process

Repeat the process for several iterations (epochs), updating weights and biases until the error converges to a minimal value, or until a predefined number of epochs is reached.

**Source code:**

import numpy as np

w = np.array([0,0,0,0])

eta = int(1)

theta = int(0)

inputs = np.array([[1,1,1,1],

[1,-1,1,1],

[1,1,-1,1],

[1,1,1,-1]])

target = [1,-1,-1,-1]

print("Inputs", inputs)

print("Target", target)

print("Weight", w)

flag = True

epoch = int(1)

while(flag):

print("Epoch",epoch)

epoch +=1

flag = False

for i in range(inputs.shape[0]):

print("For input ",i)

net\_input = sum(inputs[i,:]\*w)

print("net\_input",net\_input)

if(net\_input > theta):

y\_out = 1

elif(net\_input>= -theta and net\_input <= theta):

y\_out = 0

else:

y\_out = -1

print("Y\_out", y\_out)

if(y\_out != target[i]):

flag = True

w[:]+=eta\*target[i]\*inputs[i,:]

print("Updated w", w)

**Output:**

Inputs [[ 1 1 1 1]

[ 1 -1 1 1]

[ 1 1 -1 1]

[ 1 1 1 -1]]

Target [1, -1, -1, -1]

Weight [0 0 0 0]

Epoch 1

For input 0

net\_input 0

Y\_out 0

Updated w [1 1 1 1]

For input 1

net\_input 2

Y\_out 1

Updated w [0 2 0 0]

For input 2

net\_input 2

Y\_out 1

Updated w [-1 1 1 -1]

For input 3

net\_input 2

Y\_out 1

Updated w [-2 0 0 0]

Epoch 2

For input 0

net\_input -2

Y\_out -1

Updated w [-1 1 1 1]

For input 1

net\_input 0

Y\_out 0

Updated w [-2 2 0 0]

For input 2

net\_input 0

Y\_out 0

Updated w [-3 1 1 -1]

For input 3

net\_input 0

Y\_out 0

Updated w [-4 0 0 0]

Epoch 3

For input 0

net\_input -4

Y\_out -1

Updated w [-3 1 1 1]

For input 1

net\_input -2

Y\_out -1

For input 2

net\_input -2

Y\_out -1

For input 3

net\_input -2

Y\_out -1

Epoch 4

For input 0

net\_input 0

Y\_out 0

Updated w [-2 2 2 2]

For input 1

net\_input 0

Y\_out 0

Updated w [-3 3 1 1]

For input 2

net\_input 0

Y\_out 0

Updated w [-4 2 2 0]

For input 3

net\_input 0

Y\_out 0

Updated w [-5 1 1 1]

Epoch 5

For input 0

net\_input -2

Y\_out -1

Updated w [-4 2 2 2]

For input 1

net\_input -2

Y\_out -1

For input 2

net\_input -2

Y\_out -1

For input 3

net\_input -2

Y\_out -1

Epoch 6

For input 0

net\_input 2

Y\_out 1

For input 1

net\_input -2

Y\_out -1

For input 2

net\_input -2

Y\_out -1

For input 3

net\_input -2

Y\_out -1

**Discussion:**

This code implements a simple Perceptron algorithm for binary classification using a sign activation function. The weights are initially set to zero, and the learning rate is 1. The input vectors, which include a bias term, are processed in a loop for multiple epochs. The training continues until no more weight updates are needed, indicating convergence.