NAME: Renu Tamsekar

ROLL NO: 3253

TITLE: PERCEPTRON

LAB NO: 1

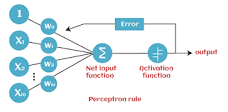
PROBLEM STATEMENT :  Write a program to implement a Perceptron Learning Algorithm

METHODOLOGY :

* Required libraries are imported.
* A unit step function is defined in which when the argument passed is greater than or equal to 0, i.e. positive then it will return 1, else it will return 0.
* Then a perceptron model is designed with 3 input parameters: input, weight and bias. The intermediate output is the dot product of the input and the weight and that is added with the bias and then this intermediate output is then given to the unit step function.
* This model is then tested on an AND gate where the bias is -1.5 and on an OR gate where the bias is -0.5.
* Further this model is also tested on NAND gate (bias=1.5) and an XOR gate (bias=0.5) where the weights are initialized to w1=-1 and w2=-1

APPLICATIONS:

* Multilayer perceptron neural networks are commonly used by different organizations to encode databases, points of entry, monitor access data, and routinely check the consistency of the database security.
* to predict the class of a sample.
* Multilayer Perceptrons are widely used to solve problems requiring supervised learning and research into computational neuroscience and parallel distributed processing.



RESULTS :

For the AND gate, using this w1\*x1+w2\*x2+b equation and w1=w2=1 and b=-1.5 we get the :

* 1st row when x1=0, x2=0 : 0+0-1.5=-1.5 ---0
* 2nd row when x1=0, x2=1: 0+1-1.5=-0.5 ---0
* 3rd row when x1=1, x2=0:1+0-1.5=-0.5 ---0
* 4th row when x1=1, x2=1:1+1-1.5=1.5 ---1

For the OR gate, using this w1\*x1+w2\*x2+b equation and w1=w2=1 and b=-0.5 we get the :

* 1st row when x1=0, x2=0 : 0+0-0.5=-0.5 ---0
* 2nd row when x1=0, x2=1: 0+1-0.5=0.5---1
* 3rd row when x1=1, x2=0:1+0-0.5=0.5 ---1
* 4th row when x1=1, x2=1:1+1-0.5=1.5---1

For the NAND gate, using this w1\*x1+w2\*x2+b equation and w1=w2=-1 and b=0.5 we get the :

* 1st row when x1=0, x2=0 : 0+0+0.5=0.5 ---1
* 2nd row when x1=0, x2=1: 0-1+0.5=-0.5---1
* 3rd row when x1=1, x2=0:-1+0+0.5=-0.5 ---1
* 4th row when x1=1, x2=1:-1-1+0.5=-1.5---0

For the XOR gate, using this w1\*x1+w2\*x2+b equation and w1=w2=-1 and b=1.5 we get the :

* 1st row when x1=0, x2=0 : 0+0+1.5=1.5 ---0
* 2nd row when x1=0, x2=1: 0-1+1.5=0.5---1
* 3rd row when x1=1, x2=0:-1+0+1.5=0.5 ---1
* 4th row when x1=1, x2=1:-1-1+1.5=-0.5---0

OBSERVATION :

* These gates are therefore successfully implemented using these perceptron models.
* For AND the model is: x1+x2-1.5
* For OR the model is: x1+x2-0.5
* For NAND the model is: -x1-x2+1.5
* For AND the model is: -x1-x2+0.5

CONCLUSION :

* A neural network link that contains computations to track features and uses Artificial Intelligence in the input data is known as Perceptron. This neural links to the artificial neurons using simple logic gates with binary outputs.
* There are 2 types of perceptrons: single layer and multi layer
* Single layer perceptrons - These can only learn linearly separable patterns.
* Multilayer perceptrons - These have the greatest processing power. They are a class of feedforward neural networks. They have a hidden layer and use sophisticated algorithms like backpropagation.

PRINT OF CODE AND OUTPUT

# importing Python library

import numpy as np

# define Unit Step Function\* Activation function

def unitStep(v):

    if v >= 0:

        return 1

    else:

        return 0

# design Perceptron Model

def perceptronModel(x, w, b):

    v = np.dot(w, x) + b

    y = unitStep(v)

    return y

# AND Logic Function

# w1 = 1, w2 = 1, b = -1.5

def AND\_logicFunction(x):

    w = np.array([1, 1])

    b = -1.5

    return perceptronModel(x, w, b)

# testing the Perceptron Model

test1 = np.array([0, 0])

test2 = np.array([0, 1])

test3 = np.array([1, 0])

test4 = np.array([1, 1])

print("AND(0, 0) =",AND\_logicFunction(test1))

print("AND(0, 1) =",AND\_logicFunction(test2))

print("AND(1, 0) =",AND\_logicFunction(test3))

print("AND(1, 1) =",AND\_logicFunction(test4))

#print("AND({}, {}) = {}".format(0, 0, AND\_logicFunction(test1)))

#print("AND({}, {}) = {}".format(0, 1, AND\_logicFunction(test2)))

#print("AND({}, {}) = {}".format(1, 0, AND\_logicFunction(test3)))

#print("AND({}, {}) = {}".format(1, 1, AND\_logicFunction(test4)))

AND(0, 0) = 0

AND(0, 1) = 0

AND(1, 0) = 0

AND(1, 1) = 1

def OR\_logicFunction(x):

    w = np.array([1, 1])

    b = -0.5

    return perceptronModel(x, w, b)

# testing the Perceptron Model

test1 = np.array([0, 0])

test2 = np.array([0, 1])

test3 = np.array([1, 0])

test4 = np.array([1, 1])

print("OR(0, 0) =",OR\_logicFunction(test1))

print("OR(0, 1) =",OR\_logicFunction(test2))

print("OR(1, 0) =",OR\_logicFunction(test3))

print("OR(1, 1) =",OR\_logicFunction(test4))

#print("OR({}, {}) = {}".format(0, 1, OR\_logicFunction(test1)))

#print("OR({}, {}) = {}".format(0, 1, OR\_logicFunction(test2)))

#print("OR({}, {}) = {}".format(1, 0, OR\_logicFunction(test3)))

#print("OR({}, {}) = {}".format(1, 1, OR\_logicFunction(test4)))

OR(0, 0) = 0

OR(0, 1) = 1

OR(1, 0) = 1

OR(1, 1) = 1

# NAND Logic Function

# w1 =-1, w2 = -1, b = 1.5

def NAND\_logicFunction(x):

    w = np.array([-1, -1])

    b = 1.5

    return perceptronModel(x, w, b)

# testing the Perceptron Model

test1 = np.array([0, 0])

test2 = np.array([0, 1])

test3 = np.array([1, 0])

test4 = np.array([1, 1])

print("NAND({}, {}) = {}".format(0, 0, NAND\_logicFunction(test1)))

print("NAND({}, {}) = {}".format(0, 1, NAND\_logicFunction(test2)))

print("NAND({}, {}) = {}".format(1, 0, NAND\_logicFunction(test3)))

print("NAND({}, {}) = {}".format(1, 1, NAND\_logicFunction(test4)))

NAND(0, 0) = 1

NAND(0, 1) = 1

NAND(1, 0) = 1

NAND(1, 1) = 0

def XOR\_logicFunction(x):

    w = np.array([1, 1])

    b = -1.5

    h1=OR\_logicFunction(x)

    h2=NAND\_logicFunction(x)

    final\_x=np.array([h1, h2])

    final\_output=AND\_logicFunction(final\_x)

    return final\_output

test1 = np.array([0, 0])

test2 = np.array([0, 1])

test3 = np.array([1, 0])

test4 = np.array([1, 1])

print("XOR({}, {}) = {}".format(0, 0, XOR\_logicFunction(test1)))

print("XOR({}, {}) = {}".format(0, 1, XOR\_logicFunction(test2)))

print("XOR({}, {}) = {}".format(1, 0, XOR\_logicFunction(test3)))

print("XOR({}, {}) = {}".format(1, 1, XOR\_logicFunction(test4)))

XOR(0, 0) = 0

XOR(0, 1) = 1

XOR(1, 0) = 1

XOR(1, 1) = 0