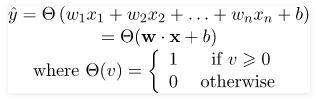
**Practical no-12 :To implement NAND logic gate**

**using Perceptron algorithm .**

In the field of Machine Learning, the Perceptron is a Supervised Learning Algorithm for binary classifiers. The Perceptron Model implements the following function:



For a particular choice of the weight vector x and bias parameter b, the model predicts output y(cap) for the corresponding input vector x.

**NAND** logical function truth table for ***2-bit binary variables***, i.e, the input vector x: (x1,x2) and the corresponding output y –

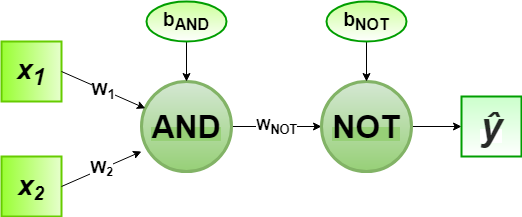
| X1 | X2 | y |
| --- | --- | --- |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

We can observe that, NAND(x1,x2) = NOT (AND (x1,x2))  
Now for the corresponding weight vector w: (w1,w2) of the input vector x: (x1,x2)  to the AND node, the associated Perceptron Function can be defined as:

Y(cap)I = Θ(w1x1+w2x2+ band)

Later on, the output of AND node Y(cap)I is the input to the NOT node with weight wnot. Then the corresponding output Y(cap) is the final output of the NAND logic function and the associated Perceptron Function can be defined as:

Y(cap) = Θ(wnotY(cap)I + bnot)



For the implementation, considered weight parameters are w1= 1, w2= 1, wnot =-1 and the bias parameters are band=-1.5, bnot = 0.5.

**Code:**

# Define Unit Step Function

def unitStep(v):

if v >= 0:

return 1

else:

return 0

# Design Perceptron Model

def perceptronModel(x, w, b):

# Calculate the dot product manually

v = sum(wi \* xi for wi, xi in zip(w, x)) + b

y = unitStep(v)

return y

# NOT Logic Function

# wNOT = -1, bNOT = 0.5

def NOT\_logicFunction(x):

wNOT = -1

bNOT = 0.5

return perceptronModel([x], [wNOT], bNOT) # Single input as a list

# AND Logic Function

# w1 = 1, w2 = 1, bAND = -1.5

def AND\_logicFunction(x):

w = [1, 1]

bAND = -1.5

return perceptronModel(x, w, bAND)

# NAND Logic Function

# with AND and NOT

# Function calls in sequence

def NAND\_logicFunction(x):

output\_AND = AND\_logicFunction(x)

output\_NOT = NOT\_logicFunction(output\_AND)

return output\_NOT

# Testing the Perceptron Model

test1 = [0, 1]

test2 = [1, 1]

test3 = [0, 0]

test4 = [1, 0]

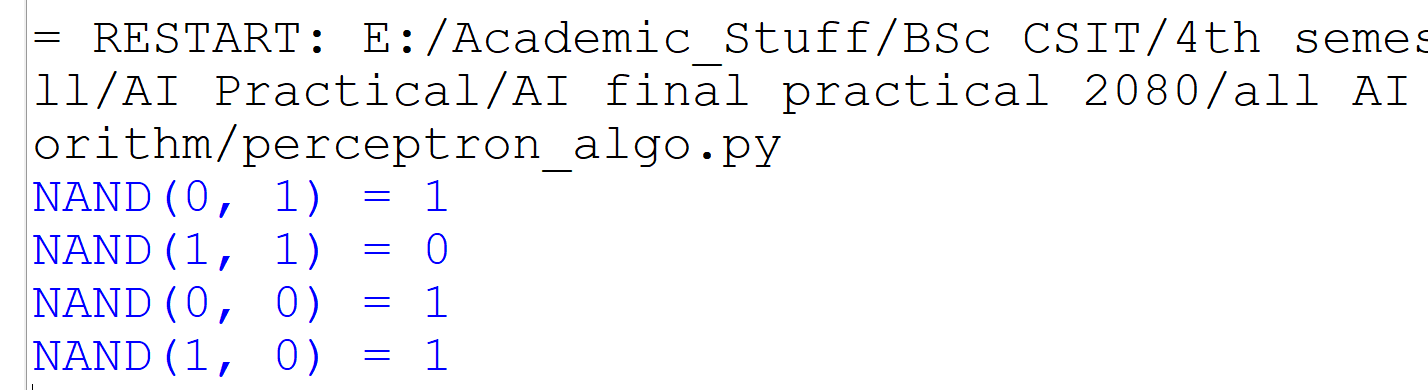
print("NAND({}, {}) = {}".format(0, 1, NAND\_logicFunction(test1)))

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

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

print("NAND({}, {}) = {}".format(1, 0, NAND\_logicFunction(test4)))

**Output:**



**Conclusion:**

Here, the model predicted output (ycap) for each of the test inputs are exactly matched with the NAND logic gate conventional output (y) according to the truth table for 2-bit binary input.  
Hence, it is verified that the perceptron algorithm for NAND logic gate is correctly implemented.