**PROGRAM ON MCCULLOCH PITTS**

class MP\_Neuron:

# firing threshold for the neuron

threshold = 0

# weights for the neuron

w1 = 0

w2 = 0

possible\_w1\_vals = [-1, 1]

possible\_w2\_vals = [-1, 1]

possible\_thresh\_vals = [-2, -1, 0, 1, 2]

def \_\_init\_\_(self, input\_matrix):

'''

Example input matrix for AND gate

| x1 | x2 | y |

| -1 | -1 | 0 |

| -1 | +1 | 0 |

| +1 | -1 | 0 |

| +1 | +1 | 1 |

'''

self.input\_matrix = input\_matrix

def iterate\_all\_values(self):

for w1 in self.possible\_w1\_vals:

self.w1 = w1

for w2 in self.possible\_w2\_vals:

self.w2 = w2

for threshold in self.possible\_thresh\_vals:

self.threshold = threshold

if self.check\_combination():

return True

return False

def check\_combination(self):

valid = True

for (x1, x2, y) in self.input\_matrix:

if not self.compare\_target(x1, x2, y):

valid = False

return valid

def compare\_target(self, x1, x2, target):

if self.neuron\_activate(x1, x2) == target:

return True

else:

return False

def neuron\_activate(self, x1, x2):

output = self.w1\*x1 + self.w2\*x2

if output >= self.threshold:

return 1

else:

return 0

if \_\_name\_\_=="\_\_main\_\_":

AND\_Matrix = [

[-1, -1, 0],

[-1, 1, 0],

[ 1, -1, 0],

[ 1, 1, 1],

]

OR\_Matrix = [

[-1, -1, 0],

[-1, 1, 1],

[ 1, -1, 1],

[ 1, 1, 1],

]

NAND\_Matrix = [

[-1, -1, 1],

[-1, 1, 1],

[ 1, -1, 1],

[ 1, 1, 0],

]

XOR\_Matrix = [

[-1, -1, 0],

[-1, 1, 1],

[ 1, -1, 1],

[ 1, 1, 0],

]

def neuron\_calculate(mp):

if mp.iterate\_all\_values():

print("Weights are : {}, {}".format(mp.w1, mp.w2))

print("Threshold is {}".format(mp.threshold))

else:

print("Not linearly separable.")

print()

print("++ AND Gate ++")

mp\_AND = MP\_Neuron(AND\_Matrix)

neuron\_calculate(mp\_AND)

print("++ OR Gate ++")

mp\_OR = MP\_Neuron(OR\_Matrix)

neuron\_calculate(mp\_OR)

print("++ NAND Gate ++")

mp\_NAND = MP\_Neuron(NAND\_Matrix)

neuron\_calculate(mp\_NAND)

print("++ XOR Gate ++")

mp\_XOR = MP\_Neuron(XOR\_Matrix)

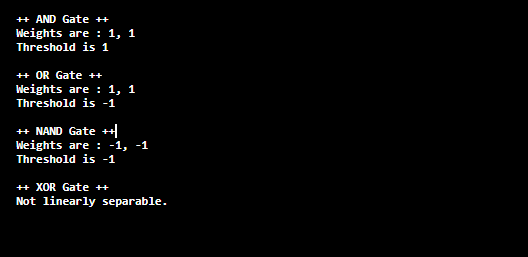
neuron\_calculate(mp\_XOR)

OUTPUT:

NOTE:

The above Python script can be executed through Google Colab and Jdoodle Python.

OUTPUT:



**PROGRAM ON TIC-TAC-TOE- GAME**

#Implementation of Two Player Tic-Tac-Toe game in Python.

''' We will make the board using dictionary

in which keys will be the location(i.e : top-left,mid-right,etc.)

and initialliy it's values will be empty space and then after every move

we will change the value according to player's choice of move. '''

theBoard = {'7': ' ' , '8': ' ' , '9': ' ' ,

'4': ' ' , '5': ' ' , '6': ' ' ,

'1': ' ' , '2': ' ' , '3': ' ' }

board\_keys = []

for key in theBoard:

board\_keys.append(key)

''' We will have to print the updated board after every move in the game and

thus we will make a function in which we'll define the printBoard function

so that we can easily print the board everytime by calling this function. '''

def printBoard(board):

print(board['7'] + '|' + board['8'] + '|' + board['9'])

print('-+-+-')

print(board['4'] + '|' + board['5'] + '|' + board['6'])

print('-+-+-')

print(board['1'] + '|' + board['2'] + '|' + board['3'])

# Now we'll write the main function which has all the gameplay functionality.

def game():

turn = 'X'

count = 0

for i in range(10):

printBoard(theBoard)

print("It's your turn," + turn + ".Move to which place?")

move = input()

if theBoard[move] == ' ':

theBoard[move] = turn

count += 1

else:

print("That place is already filled.\nMove to which place?")

continue

# Now we will check if player X or O has won,for every move after 5 moves.

if count >= 5:

if theBoard['7'] == theBoard['8'] == theBoard['9'] != ' ': # across the top

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['4'] == theBoard['5'] == theBoard['6'] != ' ': # across the middle

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['2'] == theBoard['3'] != ' ': # across the bottom

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['4'] == theBoard['7'] != ' ': # down the left side

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['2'] == theBoard['5'] == theBoard['8'] != ' ': # down the middle

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['3'] == theBoard['6'] == theBoard['9'] != ' ': # down the right side

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['7'] == theBoard['5'] == theBoard['3'] != ' ': # diagonal

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

elif theBoard['1'] == theBoard['5'] == theBoard['9'] != ' ': # diagonal

printBoard(theBoard)

print("\nGame Over.\n")

print(" \*\*\*\* " +turn + " won. \*\*\*\*")

break

# If neither X nor O wins and the board is full, we'll declare the result as 'tie'.

if count == 9:

print("\nGame Over.\n")

print("It's a Tie!!")

# Now we have to change the player after every move.

if turn =='X':

turn = 'O'

else:

turn = 'X'

# Now we will ask if player wants to restart the game or not.

restart = input("Do want to play Again?(y/n)")

if restart == "y" or restart == "Y":

for key in board\_keys:

theBoard[key] = " "

game()

if \_\_name\_\_ == "\_\_main\_\_":

game()

NOTE: EXECUTED USING GOOGLE COLAB

OUTPUT:

