Date of Performance :

Date of Submission :

EXPERIMENT NUMBER: 2

Aim: Implement Perceptron algorithm to simulate AND gate.

Objective:

To implement basic neural network models for simulating logic gate.

Software Used :Python

Theory:

Perceptron algorithm to simulate AND gate

The Perceptron algorithm is used to model the decision-making process of a logic gate by using a set of weights, bias, and a threshold to produce a binary output based on the inputs.

The Perceptron algorithm takes in two binary inputs and uses a set of weights and a bias to produce a single binary output. The inputs are multiplied by the weights and summed with the bias to produce a scalar value. This value is then compared to a threshold, and if it is above the threshold, the output is 1 (True), and if it is below the threshold, the output is 0 (False).

In the case of simulating an AND gate, the weights and bias are chosen such that if both inputs are 1, the output is 1 (True), and if either input is 0 (False), the output is 0 (False). The Perceptron algorithm can be used to simulate other logic gates, such as OR and NOT, by choosing appropriate values for the weights and bias.

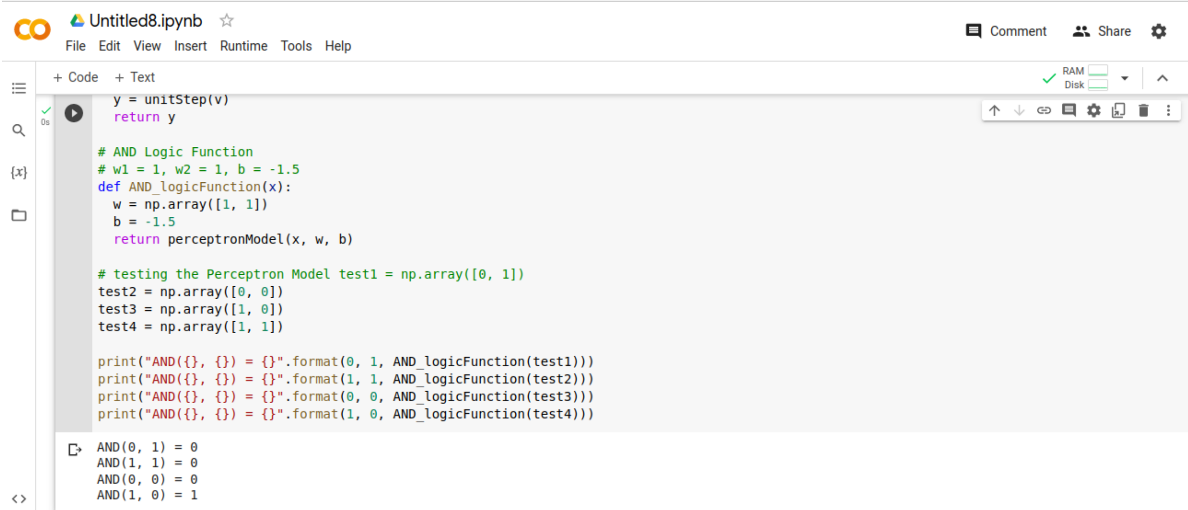
Algorithm:
Step 1: Define unit step function
Step 2: Assume w and b value
Step 3: Find net value using wx+b
Step 4: Find output value by using unit step function.
Step 5: Find error between actual and desired.
Step 6: If error is not equal to 0 update weight and bias value and go to step 5, if error is zero, go to the next step.
Step 7: Perception model is ready for further testing.

Program:

*********Perceptron algorithm to simulate AND gate***************** #
importing Python library

```
import numpy as np

# define Unit Step 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, 1])
test2 = np.array([0, 0])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```

Output:



```
  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, 1])
test2 = np.array([0, 0])
test3 = np.array([1, 0])
test4 = np.array([1, 1])

print("AND({}, {}) = {}".format(0, 1, AND_logicFunction(test1)))
print("AND({}, {}) = {}".format(1, 1, AND_logicFunction(test2)))
print("AND({}, {}) = {}".format(0, 0, AND_logicFunction(test3)))
print("AND({}, {}) = {}".format(1, 0, AND_logicFunction(test4)))
```

```
AND(0, 1) = 0
AND(1, 1) = 0
AND(0, 0) = 0
AND(1, 0) = 1
```

Conclusion/Outcome:

Thus we have implemented the Perceptron algorithm to simulate AND gate.

We also understood how to implement basic neural network models for simulating logic gate.

Marks & Signature:

R1 (4 Marks)	R2 (4 Marks)	R3 (4 Marks)	R4 (3 Mark)	Total (15 Marks)	Signature