Experiment 2

Aim: Implement AND, OR, NOR, NAND, XOR, and XNOR using ANN

Tools Used: Python

Theory: The objective is to design an artificial neural network (ANN) capable of learning and performing basic logical operations. By training the ANN on different input combinations, the model learns to replicate the behavior of each gate, demonstrating how neural networks can approximate logical functions and highlighting the versatility of ANNs in solving diverse computational tasks.

Code:

```
import numpy as np

def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  return unitStep(v)

def AND_logicFunction(x):
  w = np.array([1, 1])
  b = -1.5
  return perceptronModel(x, w, b)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

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(tes
t4)))

import numpy as np

def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  return unitStep(v)

def OR_logicFunction(x):
  w = np.array([1, 1])
  b = -0.5
  return perceptronModel(x, w, b)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("OR({}, {}) = {}".format(0, 1, OR_logicFunction(test
1)))
print("OR({}, {}) = {}".format(1, 1, OR_logicFunction(test
2)))
print("OR({}, {}) = {}".format(0, 0, OR_logicFunction(test
3)))
print("OR({}, {}) = {}".format(1, 0, OR_logicFunction(test
4)))

import numpy as np

def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
```

```
  return unitStep(v)

def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)

def OR_logicFunction(x):
  w = np.array([1, 1])
  bOR = -0.5
  return perceptronModel(x, w, bOR)

def NOR_logicFunction(x):
  output_OR = OR_logicFunction(x)
  return NOT_logicFunction(output_OR)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("NOR({}, {}) = {}".format(0, 1, NOR_logicFunction(tes
t1)))
print("NOR({}, {}) = {}".format(1, 1, NOR_logicFunction(tes
t2)))
print("NOR({}, {}) = {}".format(0, 0, NOR_logicFunction(tes
t3)))
print("NOR({}, {}) = {}".format(1, 0, NOR_logicFunction(tes
t4)))

import numpy as np

def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  return unitStep(v)
```

```
def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)

def AND_logicFunction(x):
  w = np.array([1, 1])
  bAND = -1.5
  return perceptronModel(x, w, bAND)

def NAND_logicFunction(x):
  output_AND = AND_logicFunction(x)
  return NOT_logicFunction(output_AND)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("NAND({}, {}) = {}".format(0, 1, NAND_logicFunction(t
est1)))
print("NAND({}, {}) = {}".format(1, 1, NAND_logicFunction(t
est2)))
print("NAND({}, {}) = {}".format(0, 0, NAND_logicFunction(t
est3)))
print("NAND({}, {}) = {}".format(1, 0, NAND_logicFunction(t
est4)))

import numpy as np

def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  return unitStep(v)
```

```
def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)

def AND_logicFunction(x):
  w = np.array([1, 1])
  bAND = -1.5
  return perceptronModel(x, w, bAND)

def OR_logicFunction(x):
  w = np.array([1, 1])
  bOR = -0.5
  return perceptronModel(x, w, bOR)

def XOR_logicFunction(x):
  y1 = AND_logicFunction(x)
  y2 = OR_logicFunction(x)
  y3 = NOT_logicFunction(y1)
  final_x = np.array([y2, y3])
  return AND_logicFunction(final_x)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XOR({}, {}) = {}".format(0, 1, XOR_logicFunction(tes
t1)))
print("XOR({}, {}) = {}".format(1, 1, XOR_logicFunction(tes
t2)))
print("XOR({}, {}) = {}".format(0, 0, XOR_logicFunction(tes
t3)))
print("XOR({}, {}) = {}".format(1, 0, XOR_logicFunction(tes
t4)))

import numpy as np
```

```
def unitStep(v):
  return 1 if v >= 0 else 0

def perceptronModel(x, w, b):
  v = np.dot(w, x) + b
  return unitStep(v)

def NOT_logicFunction(x):
  wNOT = -1
  bNOT = 0.5
  return perceptronModel(x, wNOT, bNOT)

def AND_logicFunction(x):
  w = np.array([1, 1])
  bAND = -1.5
  return perceptronModel(x, w, bAND)

def OR_logicFunction(x):
  w = np.array([1, 1])
  bOR = -0.5
  return perceptronModel(x, w, bOR)

def XNOR_logicFunction(x):
  y1 = OR_logicFunction(x)
  y2 = AND_logicFunction(x)
  y3 = NOT_logicFunction(y1)
  final_x = np.array([y2, y3])
  return OR_logicFunction(final_x)

test1 = np.array([0, 1])
test2 = np.array([1, 1])
test3 = np.array([0, 0])
test4 = np.array([1, 0])

print("XNOR({}, {}) = {}".format(0, 1, XNOR_logicFunction(t
est1)))
print("XNOR({}, {}) = {}".format(1, 1, XNOR_logicFunction(t
est2)))
```

```
print("XNOR({}, {}) = {}".format(0, 0, XNOR_logicFunction(test3)))
print("XNOR({}, {}) = {}".format(1, 0, XNOR_logicFunction(test4)))
```

Output:

```
AND(0, 1) = 0
AND(1, 1) = 1
AND(0, 0) = 0
AND(1, 0) = 0
OR(0, 1) = 1
OR(1, 1) = 1
OR(0, 0) = 0
OR(1, 0) = 1
NOR(0, 1) = 0
NOR(1, 1) = 0
NOR(0, 0) = 1
NOR(1, 0) = 0
NAND(0, 1) = 1
NAND(1, 1) = 0
NAND(0, 0) = 1
NAND(1, 0) = 1
XOR(0, 1) = 1
XOR(1, 1) = 0
XOR(0, 0) = 0
XOR(1, 0) = 1
XNOR(0, 1) = 0
XNOR(1, 1) = 1
XNOR(0, 0) = 1
XNOR(1, 0) = 0
```

Result: AND, OR, NOR, NAND, XOR and XNOR have been successfully implemented using ANN.

Criteria	Total Marks	Marks Obtained	Comments
Concept (A)			
Implementation (B)			
Performance (C)			
Total			