BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Rishabh Santosh Shenoy
UID no.	2023300222
Experiment No.	3

AIM:	To implement transfer/activation functions and design Artificial Neural Network (ANN)
Program 1	
PROBLEM STATEMENT :	Design and implement Artificial Neural Networks (ANN) using transfer functions like sigmoid and binary step to simulate logic gates. Build half adder and full adder circuits with neural computations, display intermediate neuron outputs, activation results, and truth tables, demonstrating how ANN models perform binary addition operations effectively.
THEORY:	Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They consist of interconnected processing units called neurons, which receive inputs, process them using mathematical functions, and produce outputs. Each neuron applies weights to the inputs and passes the result through an activation function, which determines the final output signal. ANNs are widely used in machine learning, pattern recognition, and artificial intelligence because of their ability to approximate complex nonlinear relationships. Activation or transfer functions play a crucial role in ANN design. Common functions include the sigmoid, binary step, and ReLU. The sigmoid maps inputs to values between 0 and 1, enabling probabilistic interpretations, while the binary step produces strict 0 or 1 outputs, mimicking digital logic. These functions introduce non-linearity, which is essential for solving complex tasks beyond simple linear separation. Without them, ANNs would behave like linear models, limiting their capability. By combining activation functions with appropriate weights, ANNs can simulate digital logic circuits such as AND, OR, XOR, half adders, and full adders. A half adder computes the sum and carry of two binary inputs,

whereas a full adder extends this by including a carry-in bit. Implementing these circuits using ANN concepts demonstrates how neural models replicate arithmetic operations. This approach not only validates ANN functionality but also bridges neural computation with classical digital electronics.

PROGRAM:

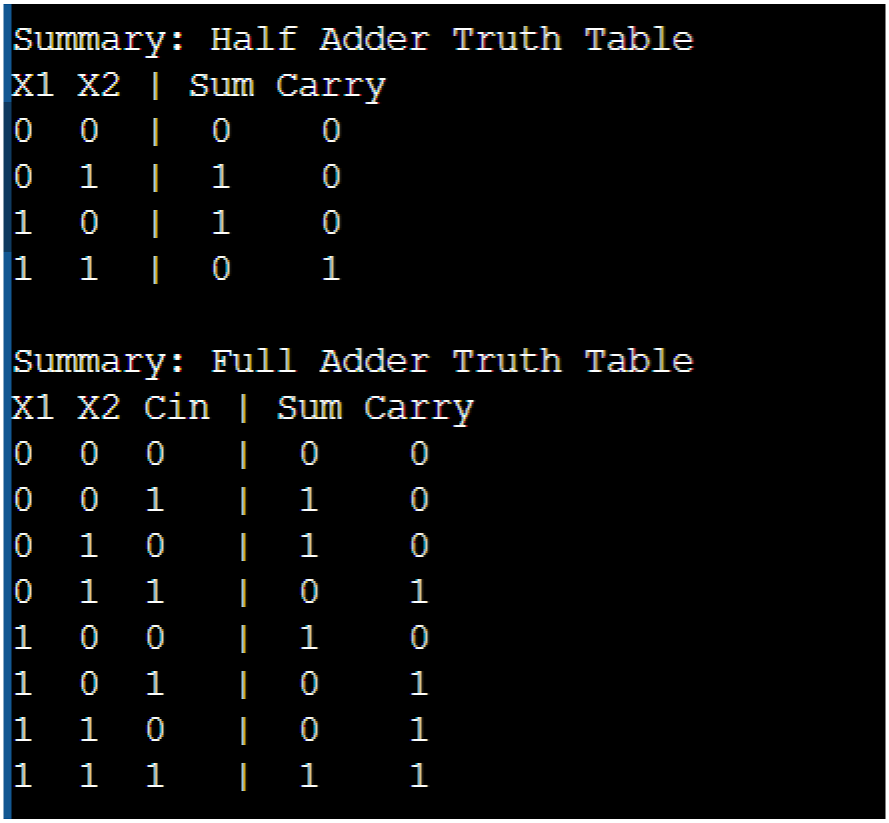
```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def binary_step(x, theta=2):
    return np.where(x >= theta, 1, 0)

def and_gate(x1, x2):
    X = np.array([x1, x2])
    W = np.array([1, 1])
    z = np.dot(X, W)
    return binary_step(z)

def half_adder(x1, x2):
    X = np.array([x1, x2])

    W_sum = np.array([1, -1])
    z_sum = np.dot(X, W_sum)
    sigmoid_sum = sigmoid(z_sum)
    sum_out = int(np.round(sigmoid_sum))

    W_carry = np.array([1, 1])
    z_carry = np.dot(X, W_carry)
    sigmoid_carry = sigmoid(z_carry)
    carry_out = int(np.round(sigmoid_carry))

    return sum_out, carry_out, z_sum, sigmoid_sum, z_carry, sigmoid_carry

def full_adder(x1, x2, cin):
    s1, c1, z1, sig1, z2, sig2 = half_adder(x1, x2)
    s2, c2, z3, sig3, z4, sig4 = half_adder(s1, cin)
    final_sum = s2
    final_carry = int(np.round(sigmoid(c1 + c2)))

    return final_sum, final_carry, z1, sig1, z2, sig2, z3, sig3, z4, sig4

half_adder_results = []
print("\nHalf Adder (Sigmoid Activation, No Bias)")
print("X1 X2 | z(Sum) | σ(z) | z(Carry) | σ(z) | Sum Carry")

for x1 in [0, 1]:
    for x2 in [0, 1]:
        s, c, z_sum, sig_sum, z_carry, sig_carry = half_adder(x1, x2)
        half_adder_results.append((x1, x2, s, c))
        print(f"{x1} {x2} | {z_sum:6.2f} | {sig_sum:.4f} | {z_carry:7.2f} | {sig_carry:.4f} | {s:^3} {c}")

full_adder_results = []
print("\nFull Adder (Sigmoid Activation, No Bias)")
print("X1 X2 Cin | z1(Sum) σ(z1) | z2(Carry) σ(z2) | z3(Sum) σ(z3) | z4(Carry) σ(z4) | Sum Carry")

for x1 in [0, 1]:
    for x2 in [0, 1]:
        for cin in [0, 1]:
            s, co, z1, sig1, z2, sig2, z3, sig3, z4, sig4 = full_adder(x1, x2, cin)
            full_adder_results.append((x1, x2, cin, s, co))
            print(f"{x1} {x2} {cin} | {z1:6.2f} {sig1:.4f} | {z2:7.2f} {sig2:.4f} | {z3:6.2f} {sig3:.4f} | {z4:7.2f} {sig4:.4f} | {s:^3} {co}")

# === Optional: Display Truth Tables Summary ===
print("\nSummary: Half Adder Truth Table")
print("X1 X2 | Sum Carry")
for x1, x2, s, c in half_adder_results:
    print(f"{x1} {x2} | {s} {c}")

print("\nSummary: Full Adder Truth Table")
print("X1 X2 Cin | Sum Carry")
```

```
for x1, x2, cin, s, c in full_adder_results:
    print(f"{x1} {x2} {cin} | {s} {c}")
```

INTERMEDIATE OUTPUT:

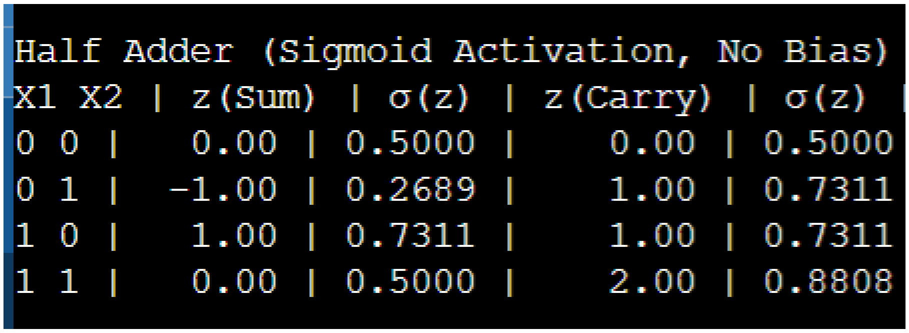
```
X1 X2 | z(XOR_approx) σ(XOR_approx) | z(AND) σ(AND) | Sum Carry
Half Adder Inputs: X1=0, X2=0
  AND -> bool=0, z=0.00, σ(z)=0.5000
  OR  -> bool=0, z=0.00, σ(z)=0.5000
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=0

0  0 |     0.0000    0.5000    |  0.00  0.5000 |   0    0
Half Adder Inputs: X1=0, X2=1
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

0  1 |     0.0000    0.5000    |  1.00  0.7311 |   1    0
Half Adder Inputs: X1=1, X2=0
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

1  0 |     0.0000    0.5000    |  1.00  0.7311 |   1    0
Half Adder Inputs: X1=1, X2=1
  AND -> bool=1, z=2.00, σ(z)=0.8808
  OR  -> bool=1, z=2.00, σ(z)=0.8808
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=1

1  1 |     0.0000    0.5000    |  2.00  0.8808 |   0    1

X1 X2 Cin | Sum Carry
Half Adder Inputs: X1=0, X2=0
  AND -> bool=0, z=0.00, σ(z)=0.5000
  OR  -> bool=0, z=0.00, σ(z)=0.5000
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=0

Half Adder Inputs: X1=0, X2=0
  AND -> bool=0, z=0.00, σ(z)=0.5000
  OR  -> bool=0, z=0.00, σ(z)=0.5000
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=0

Full Adder Inputs: X1=0, X2=0, Cin=0
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=0.00, σ=0.5000
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=0.00, �=0.5000
  Intermediate carries: c1=0, c2=0, sigmoid(c1+c2)=0.5000
  Result => Sum=0, Carry=0

0  0  0   | 0    0
Half Adder Inputs: X1=0, X2=0
  AND -> bool=0, z=0.00, σ(z)=0.5000
  OR  -> bool=0, z=0.00, σ(z)=0.5000
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=0

Half Adder Inputs: X1=0, X2=1
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Full Adder Inputs: X1=0, X2=0, Cin=1
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=0.00, σ=0.5000
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Intermediate carries: c1=0, c2=0, sigmoid(c1+c2)=0.5000
  Result => Sum=1, Carry=0

0  0  1   | 1    0
Half Adder Inputs: X1=0, X2=1
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Half Adder Inputs: X1=1, X2=0
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Full Adder Inputs: X1=0, X2=1, Cin=0
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Intermediate carries: c1=0, c2=0, sigmoid(c1+c2)=0.5000
  Result => Sum=1, Carry=0

0  1  0  | 1    0
Half Adder Inputs: X1=0, X2=1
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Half Adder Inputs: X1=1, X2=1
  AND -> bool=1, z=2.00, σ(z)=0.8808
  OR  -> bool=1, z=2.00, σ(z)=0.8808
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=1

Full Adder Inputs: X1=0, X2=1, Cin=1
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=2.00, σ=0.8808
  Intermediate carries: c1=0, c2=1, sigmoid(c1+c2)=0.7311
  Result => Sum=0, Carry=1

0  1  1  | 0    1
Half Adder Inputs: X1=1, X2=0
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Half Adder Inputs: X1=1, X2=0
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Full Adder Inputs: X1=1, X2=0, Cin=0
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Intermediate carries: c1=0, c2=0, sigmoid(c1+c2)=0.5000
  Result => Sum=1, Carry=0

1  0  0  | 1    0
Half Adder Inputs: X1=1, X2=0
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Half Adder Inputs: X1=1, X2=1
  AND -> bool=1, z=2.00, σ(z)=0.8808
  OR  -> bool=1, z=2.00, σ(z)=0.8808
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=1

Full Adder Inputs: X1=1, X2=0, Cin=1
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=2.00, σ=0.8808
  Intermediate carries: c1=0, c2=1, sigmoid(c1+c2)=0.7311
  Result => Sum=0, Carry=1

1  0  1  | 0    1
Half Adder Inputs: X1=1, X2=1
  AND -> bool=1, z=2.00, σ(z)=0.8808
  OR  -> bool=1, z=2.00, σ(z)=0.8808
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=1

Half Adder Inputs: X1=0, X2=0
  AND -> bool=0, z=0.00, σ(z)=0.5000
  OR  -> bool=0, z=0.00, σ(z)=0.5000
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=0

Full Adder Inputs: X1=1, X2=1, Cin=0
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=2.00, σ=0.8808
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=0.00, σ=0.5000
  Intermediate carries: c1=1, c2=0, sigmoid(c1+c2)=0.7311
  Result => Sum=0, Carry=1

1  1  0   | 0    1
Half Adder Inputs: X1=1, X2=1
  AND -> bool=1, z=2.00, σ(z)=0.8808
  OR  -> bool=1, z=2.00, σ(z)=0.8808
  XOR -> bool=0, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=0, Carry=1

Half Adder Inputs: X1=0, X2=1
  AND -> bool=0, z=1.00, σ(z)=0.7311
  OR  -> bool=1, z=1.00, σ(z)=0.7311
  XOR -> bool=1, approx z=0.0000, σ(approx)=0.5000
  Result => Sum=1, Carry=0

Full Adder Inputs: X1=1, X2=1, Cin=1
  Half1 Sum approx z=0.0000, σ=0.5000  Carry z=2.00, σ=0.8808
  Half2 Sum approx z=0.0000, σ=0.5000  Carry z=1.00, σ=0.7311
  Intermediate carries: c1=1, c2=0, sigmoid(c1+c2)=0.7311
  Result => Sum=1, Carry=1

1  1  1   | 1    1
```

OUTPUT:



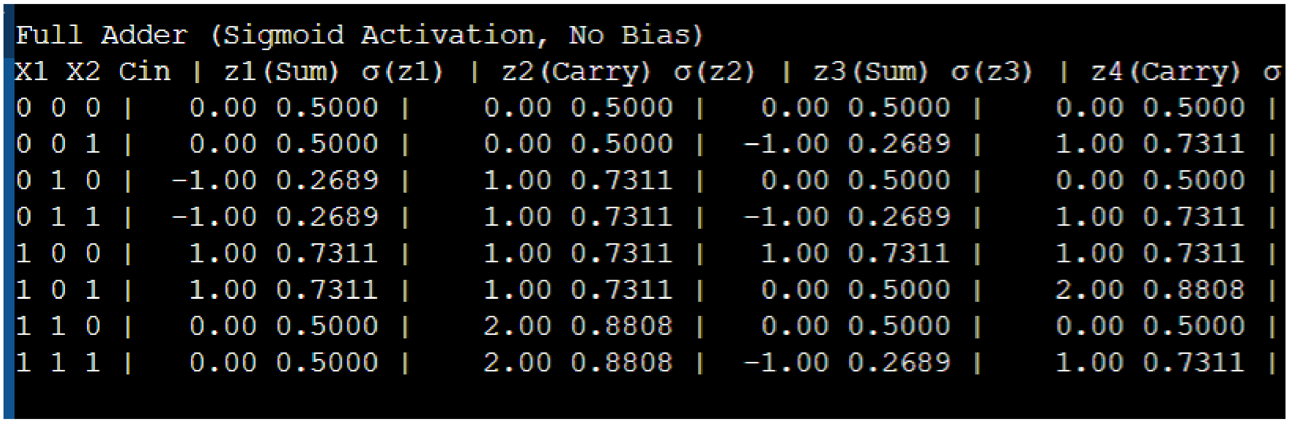
```
Summary: Half Adder Truth Table
X1 X2 | Sum Carry
0  0  |  0    0
0  1  |  1    0
1  0  |  1    0
1  1  |  0    1

Summary: Full Adder Truth Table
X1 X2 Cin | Sum Carry
0  0  0   |  0    0
0  0  1   |  1    0
0  1  0   |  1    0
0  1  1   |  0    1
1  0  0   |  1    0
1  0  1   |  0    1
1  1  0   |  0    1
1  1  1   |  1    1
```

OUTPUT 2:



```
Half Adder (Sigmoid Activation, No Bias)
X1 X2 | z(Sum) | σ(z) | z(Carry) | σ(z) |
0 0 |    0.00 | 0.5000 |     0.00 | 0.5000
0 1 |   -1.00 | 0.2689 |     1.00 | 0.7311
1 0 |    1.00 | 0.7311 |     1.00 | 0.7311
1 1 |    0.00 | 0.5000 |     2.00 | 0.8808
```

OUTPUT 3:



```
Full Adder (Sigmoid Activation, No Bias)
X1 X2 Cin | z1(Sum) σ(z1)  | z2(Carry) σ(z2) | z3(Sum) σ(z3)  | z4(Carry) σ
0 0 0 |   0.00 0.5000 |    0.00 0.5000 |   0.00 0.5000 |    0.00 0.5000 |
0 0 1 |   0.00 0.5000 |    0.00 0.5000 |  -1.00 0.2689 |    1.00 0.7311 |
0 1 0 |  -1.00 0.2689 |    1.00 0.7311 |   0.00 0.5000 |    0.00 0.5000 |
0 1 1 |  -1.00 0.2689 |    1.00 0.7311 |  -1.00 0.2689 |    1.00 0.7311 |
1 0 0 |   1.00 0.7311 |    1.00 0.7311 |   1.00 0.7311 |    1.00 0.7311 |
1 0 1 |   1.00 0.7311 |    1.00 0.7311 |   0.00 0.5000 |    2.00 0.8808 |
1 1 0 |   0.00 0.5000 |    2.00 0.8808 |   0.00 0.5000 |    0.00 0.5000 |
1 1 1 |   0.00 0.5000 |    2.00 0.8808 |  -1.00 0.2689 |    1.00 0.7311 |
```

CONCLUSION:

In this experiment, I have understood the fundamental working of the McCulloch-Pitts neuron model and its application in designing simple logic circuits such as half-adders and full-adders. By implementing the sigmoid activation function without bias, I explored how neuron outputs approximate logical operations. I observed the intermediate values of z and $\sigma(z)$ where σ(z) helped me correlate the mathematical computations with actual digital logic. This reinforced my understanding of artificial neural networks and their potential to simulate binary logic.

Furthermore, I have understood the significance of combining multiple neurons to achieve complex computations, as demonstrated in the full-adder using two half-adders. The experiment highlighted how careful weight selection influences neuron outputs and the approximation of logic functions. By analyzing both the intermediate neuron outputs and the final truth tables, I gained practical insights into ANN design, logical reasoning, and the role of activation functions in simulating digital circuits.