Assignment 1

Experiment 1 and 2

Title: ANN and Its Models

Name of Student: Sangeet Agrawal PRN No. 21070122140

DoP1: 15 Jul

DoP2: 22 Jul DoS: 28 Jul

Aim:

a) Implement a Program to Generate the Output of Logic AND Function by McCulloch–Pitts Neuron Model. The Threshold on Unit Is 2. Simulate same by changing T = 1

b) Implement a Program to Generate Output for OR Function Using McCulloch–Pitts Neurons with Threshold Value 1

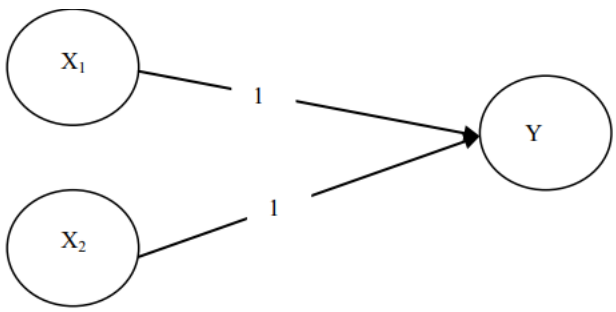
c) Implement a Program to Generate Output for XOR Function Using McCulloch–Pitts Neurons with Threshold Value 1

d) Implement a Program to Generate Output for ANDNOT Function Using McCulloch–Pitts Neurons

Learning Outcome:
1. To understand the fundamentals of ANN
2. To implement McCulloch–Pitts Neuron Model

Hardware/Software: Google Collab

Problem Statement: X1=[1 1 0 0], X2=[1 0 1 0] T=2, w1=1, w2=1



* Theory :-

→ The McCulloch-Pitts Neural Network:

It is a simple model of a neuron, developed in 1943. It takes binary inputs (0 or 1), multiplies each input by a weight, sums these values, and compares the sum to a threshold. If the sum is greater than or equal to the threshold, the neuron "fires" and outputs a 1; otherwise, it output a 0. This model helps us understand the basics of artificial neural networks.

→ McCulloch - Pitts Neuron Architecture:

i) Inputs : Binary values (0 or 1) from multiple sources.
ii) Weights : Numbers that multiply each input
iii) Summation : Adds up the weighted input.
iv) Threshold : A set value that the sum is compared to.
v) Output : Binary value (0 or 1) based on the threshold comparison.

Program a)

```
# AND Function

def mcCulloch_pitts_and_t1(inputs):
   # Adjusted weights for AND function
   weights = [0.5, 0.5]
   threshold = 1

   # calculate the weighted sum
   weighted_sum = sum(weight * input_val for weight, input_val in
zip(weights, inputs))

   # apply the threshold to get the output
   output = 1 if weighted_sum >= threshold else 0
   return output

def mcCulloch_pitts_and_t2(inputs):
   # weights for AND function
   weights = [1, 1]
   threshold = 2

   # calculate the weighted sum
   weighted_sum = sum(weight * input_val for weight, input_val in
zip(weights, inputs))

   # apply the threshold to get the output
   output = 1 if weighted_sum >= threshold else 0
   return output

# Test inputs
test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

print("\nAND Function with T = 1")
for inputs in test_inputs:
   print(f"Inputs: {inputs} -> Output:
{mcCulloch_pitts_and_t1(inputs)}")

print("AND Function with T = 2")
for inputs in test_inputs:
   print(f"Inputs: {inputs} -> Output:
{mcCulloch_pitts_and_t2(inputs)}")
```

Output Screenshot:

```
AND Function with T = 1
Inputs: (0, 0) -> Output: 0
Inputs: (0, 1) -> Output: 0
Inputs: (1, 0) -> Output: 0
Inputs: (1, 1) -> Output: 1
AND Function with T = 2
Inputs: (0, 0) -> Output: 0
Inputs: (0, 1) -> Output: 0
Inputs: (1, 0) -> Output: 0
Inputs: (1, 1) -> Output: 1
```

Program b)

```
# OR Function

def mcCulloch_pitts_or(inputs, threshold):
    # weights for OR function
    weights = [1, 1]

    # calculate the weighted sum
    weighted_sum = sum(weight * input_val for weight, input_val in
zip(weights, inputs))

    # apply the threshold to get the output
    output = 1 if weighted_sum >= threshold else 0
    return output

# Test with threshold T = 1
threshold_1 = 1
test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

print("OR Function with T = 1")
for inputs in test_inputs:
    print(f"Inputs: {inputs} -> Output: {mcCulloch_pitts_or(inputs,
threshold_1)}")
```

Output Screenshot:

```
OR Function with T = 1
Inputs: (0, 0) -> Output: 0
Inputs: (0, 1) -> Output: 1
Inputs: (1, 0) -> Output: 1
Inputs: (1, 1) -> Output: 1
```

Program: c)

```python
# XOR Function

def mcculloch_pitts_neuron(inputs, weights, threshold):
   weighted_sum = sum(w * i for w, i in zip(weights, inputs))
   return 1 if weighted_sum >= threshold else 0

def xor_mcculloch_pitts(x1, x2):
   # Neuron 1 (OR gate)
   n1_output = mcculloch_pitts_neuron([x1, x2], [1, 1], 1)

   # Neuron 2 (AND gate)
   n2_output = mcculloch_pitts_neuron([x1, x2], [1, 1], 2)

   # Neuron 3 (XOR gate)
   xor_output = mcculloch_pitts_neuron([n1_output, n2_output], [1,
-2], 1)

   return xor_output

# Test the XOR function
print("XOR Function Outputs:")
for x1 in [0, 1]:
   for x2 in [0, 1]:
       print(f"XOR({x1}, {x2}) = {xor_mcculloch_pitts(x1, x2)}")
```

Output Screenshot:

```
XOR Function Outputs:
XOR(0, 0) = 0
XOR(0, 1) = 1
XOR(1, 0) = 1
XOR(1, 1) = 0
```

Program d)

```python
# NAND Function

def mcCulloch_pitts_nand(inputs):
   # Weights for NAND function
   weights = [-1, -1]
   threshold = -0.5

   # Calculate the weighted sum
   weighted_sum = sum(weight * input_val for weight, input_val in
zip(weights, inputs))

   # Apply the threshold to get the output
   output = 1 if weighted_sum >= threshold else 0

   return output

# Test inputs
test_inputs = [(0, 0), (0, 1), (1, 0), (1, 1)]

print("\nNAND Function:")
for inputs in test_inputs:
   print(f"Inputs: {inputs} -> Output:
{mcCulloch_pitts_nand(inputs)}")
```

Output Screenshot:

```
NAND Function:
Inputs: (0, 0) -> Output: 1
Inputs: (0, 1) -> Output: 0
Inputs: (1, 0) -> Output: 0
Inputs: (1, 1) -> Output: 0
```

→ Conclusion:

The McCulloch-Pitts neuron model is a foundational concept in neural networks, demonstrating how simple binary operations can model neuron behavior. It forms the basis for understanding more complex artificial neural network structures and their functions.

Assignment 1

Experiment 3

Title: ANN and Its Models

Name of Student: Sangeet Agrawal PRN No. 21070122140

DoP: 29 Jul DoS: 4 Aug

Aim: e) Implement activation functions Binary unipolar, binary bipolar, continuous unipolar, sigmoid, and ReLU function

Learning Outcome:
1. To understand the fundamentals of ANN
2. To implement McCulloch–Pitts Neuron Model

Hardware/Software: MATLAB Online

* Theory :

Activation Function : It calculates the output of a neuron by applying a mathematical operation to the sum of its weighted inputs. They are essential for introducing non-linearity into neural networks. Neurons in the same layer use the activation function. There are linear and non-linear activation functions with non-linear being crucial for multi-layer networks.

* Types of Activation Functions:

1) Linear Activation Function:
→ Output proportional to input.

2) Binary Sigmoid (Unipolar Sigmoid)
→ Output: 0 to 1
→ Formula: $f(x) = \frac{1}{1+e^{-x}}$

3) Bipolar Sigmoid:
→ Output : −1 to 1
→ Formula: $f(x) = \frac{2}{1+e^{-x}} - 1$

HQ

4) ReLU (Rectified Linear Unit):
→ Output: 0 for negative, linear for positive
→ Formula: $f(x) = max(0, x)$

5) Leaky ReLU:
→ Allows small gradient for negative inputs
→ Formula: $f(x) = max(0.01x, x)$.

6) Unipolar Binary:
→ Output: 0 or 1

7) Bipolar Binary:
→ Output: -1 or 1

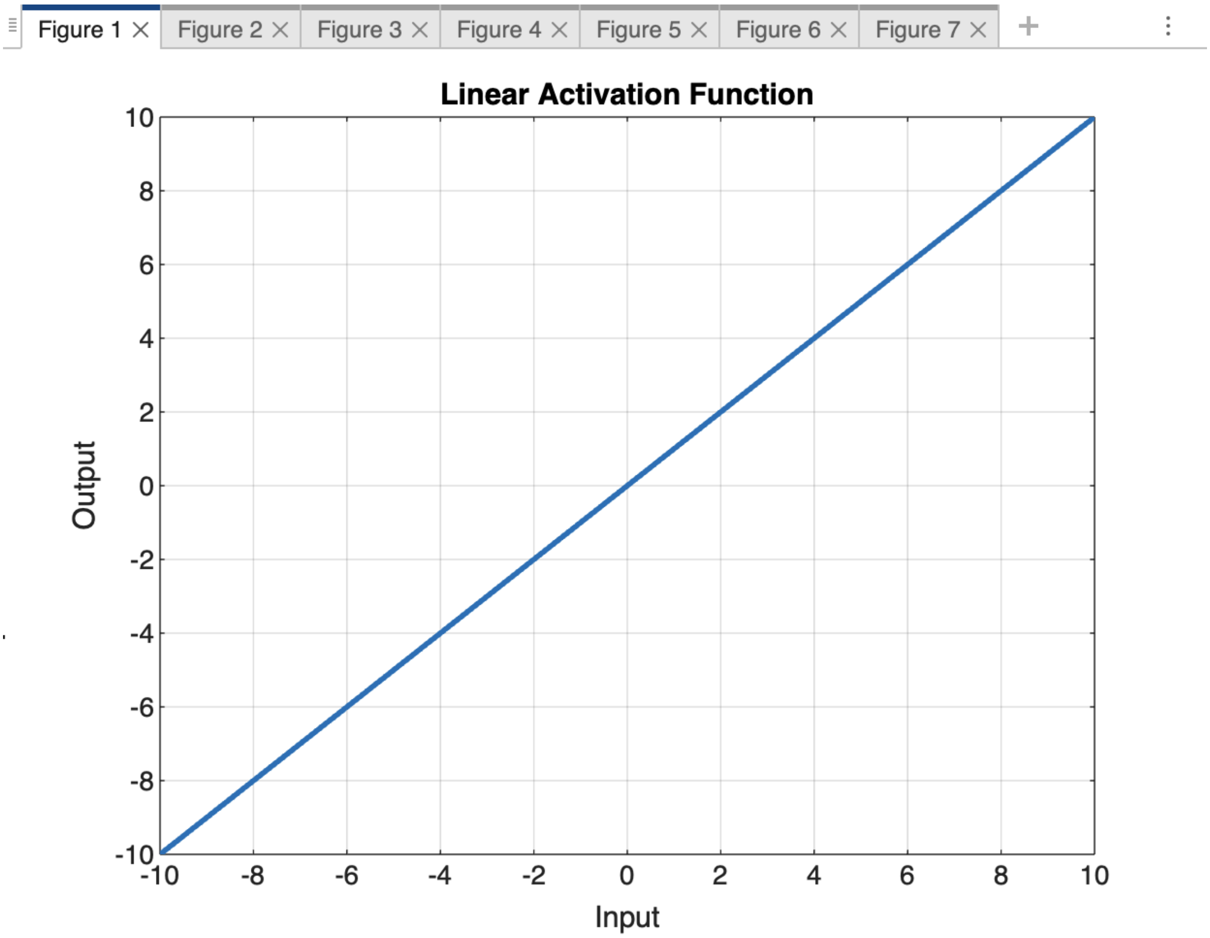
Program:

```
function Experiment_3()
   while true
       % Display menu options
       fprintf('Select an activation function to plot:\n');
       fprintf('1. Linear\n');
       fprintf('2. Binary Sigmoid\n');
       fprintf('3. Bipolar Sigmoid\n');
       fprintf('4. ReLU\n');
       fprintf('5. Leaky ReLU\n');
       fprintf('6. Unipolar Binary\n');
       fprintf('7. Bipolar Binary\n');
       fprintf('8. Exit\n');

       % Get user choice
       choice = input('Enter your choice: ');

       % Define the range of input values
       x = -10:0.1:10;

       switch choice
           case 1
               % Linear Activation Function
               linear_activation = x;
               plot_activation(x, linear_activation, 'Linear
Activation Function');

           case 2
               % Binary Sigmoid Activation Function
               binary_sigmoid_activation = 1 ./ (1 + exp(-x));
               plot_activation(x, binary_sigmoid_activation, 'Binary
Sigmoid Activation Function');

           case 3
               % Bipolar Sigmoid Activation Function
               bipolar_sigmoid_activation = (2 ./ (1 + exp(-x))) -
```

```
1;
               plot_activation(x, bipolar_sigmoid_activation,
'Bipolar Sigmoid Activation Function');

           case 4
               % ReLU Activation Function
               relu_activation = max(0, x);
               plot_activation(x, relu_activation, 'ReLU Activation
Function');

           case 5
               % Leaky ReLU Activation Function
               alpha = 0.01;
               leaky_relu_activation = max(alpha * x, x);
               plot_activation(x, leaky_relu_activation, 'Leaky ReLU
Activation Function');

           case 6
               % Unipolar Binary Activation Function
               unipolar_binary_activation = double(x >= 0);
               plot_activation(x, unipolar_binary_activation,
'Unipolar Binary Activation Function');

           case 7
               % Bipolar Binary Activation Function
               bipolar_binary_activation = double(x >= 0) * 2 - 1;
               plot_activation(x, bipolar_binary_activation,
'Bipolar Binary Activation Function');

           case 8
               % Exit
               disp('Exiting...');
               break;

           otherwise
               disp('Invalid choice. Please select a valid
option.');
       end
   end
end
function plot_activation(x, y, title_str)
   figure;
   plot(x, y, 'LineWidth', 2);
   title(title_str);
   xlabel('Input');
   ylabel('Output');
   grid on;
end
```

Output Screenshot:

```
Command Window
>> Assignment_2
Select an activation function to plot:
1. Linear
2. Binary Sigmoid
3. Bipolar Sigmoid
4. ReLU
5. Leaky ReLU
6. Unipolar Binary
7. Bipolar Binary
8. Exit
Enter your choice:
```



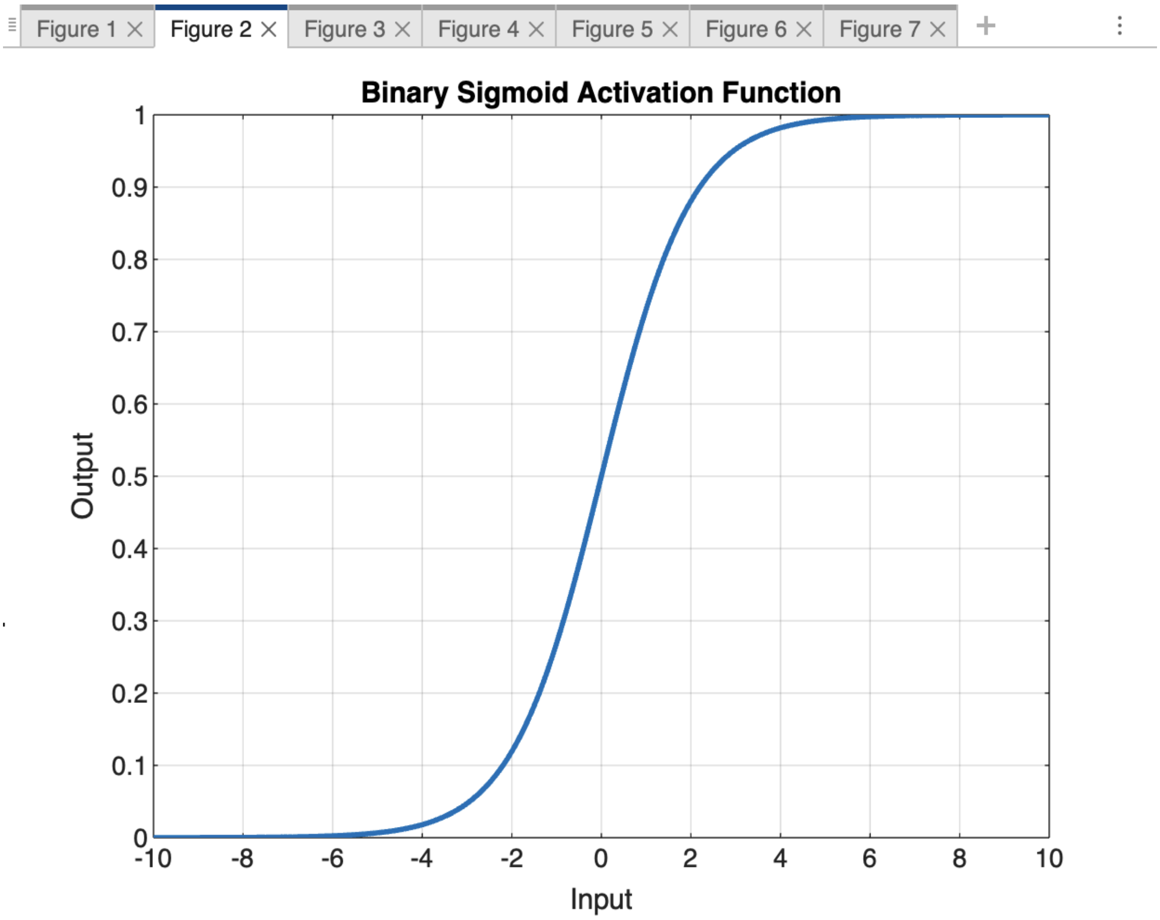


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
Binary Sigmoid Activation Function
Output
Input
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-8
-6
-4
-2
0
2
4
6
8
10

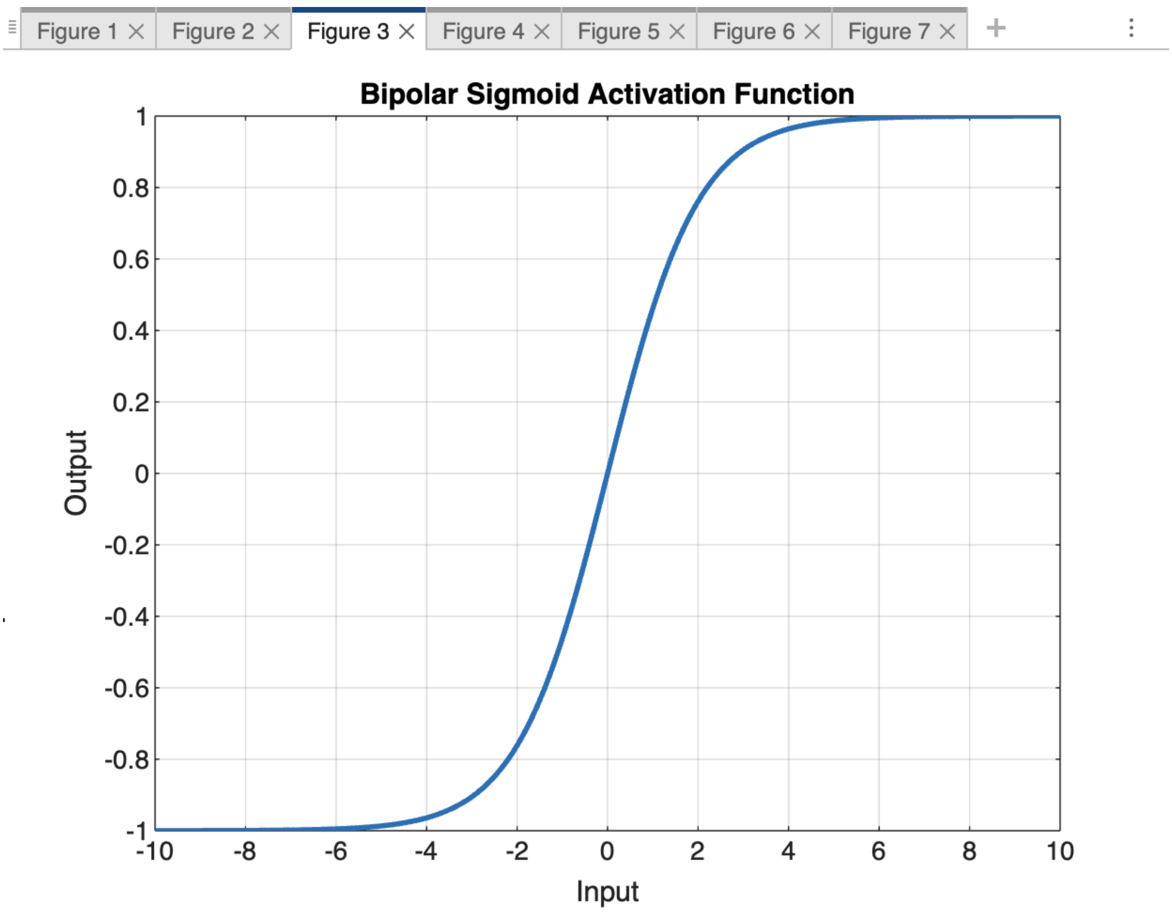


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
Bipolar Sigmoid Activation Function
Output
Input
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-10
-8
-6
-4
-2
0
2
4
6
8
10

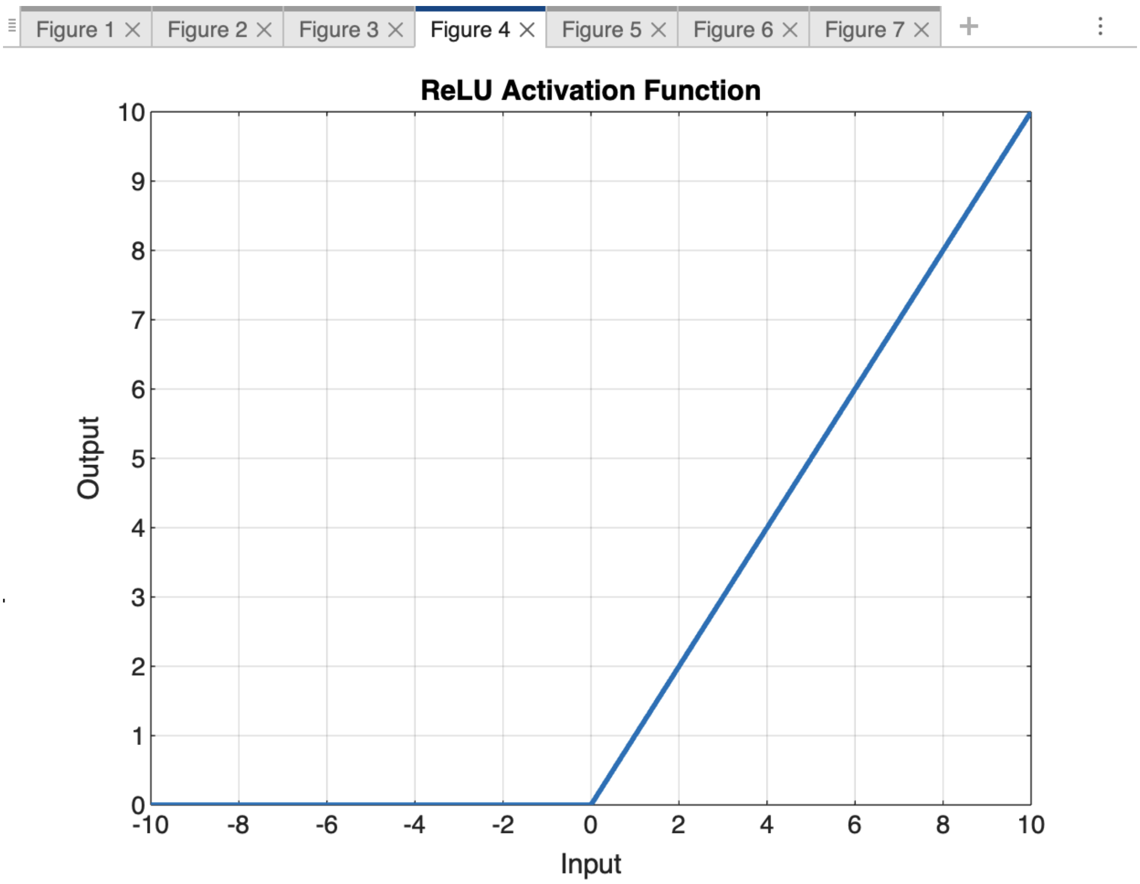


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
ReLU Activation Function
Output
Input
10
9
8
7
6
5
4
3
2
1
0
-10
-8
-6
-4
-2
0
2
4
6
8
10

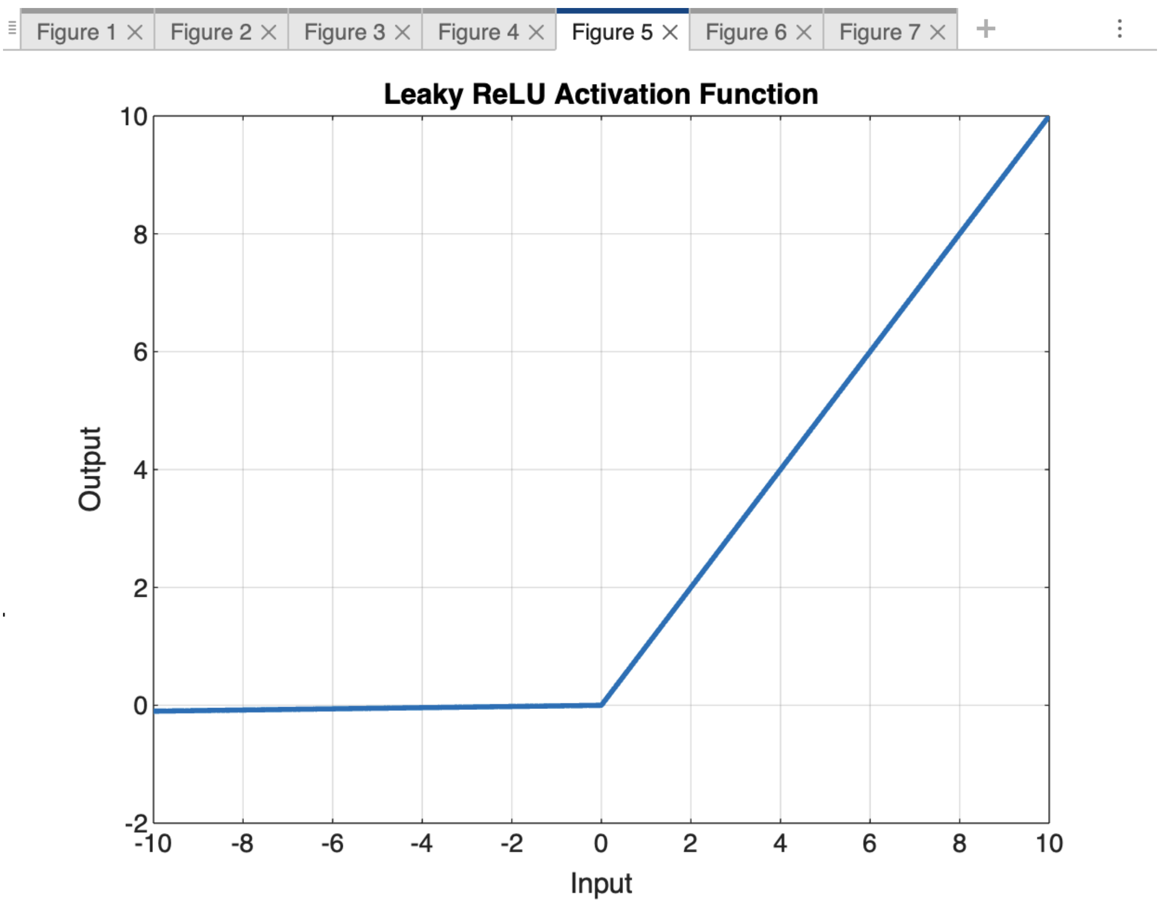


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
Leaky ReLU Activation Function
Output
Input
10
8
6
4
2
0
-2
-10
-8
-6
-4
-2
0
2
4
6
8
10

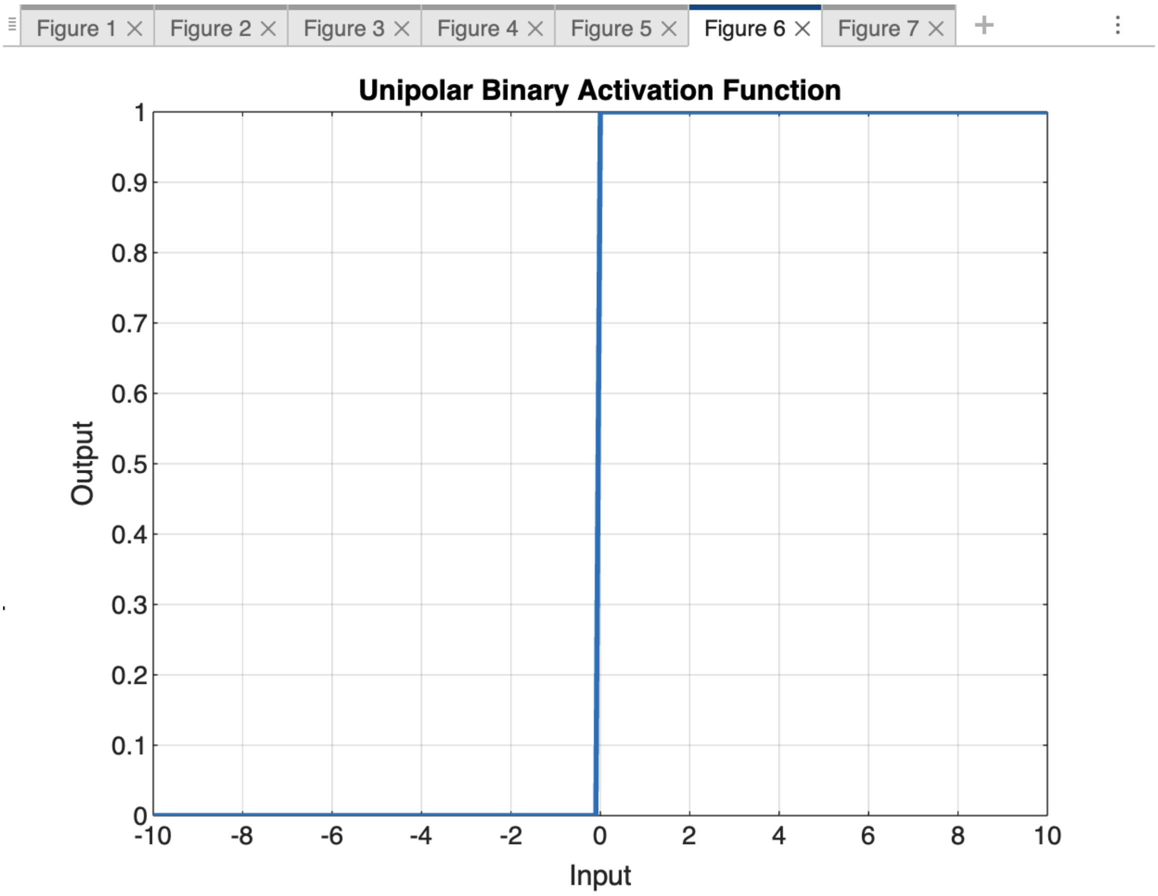


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
Unipolar Binary Activation Function
Output
Input
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-8
-6
-4
-2
0
2
4
6
8
10

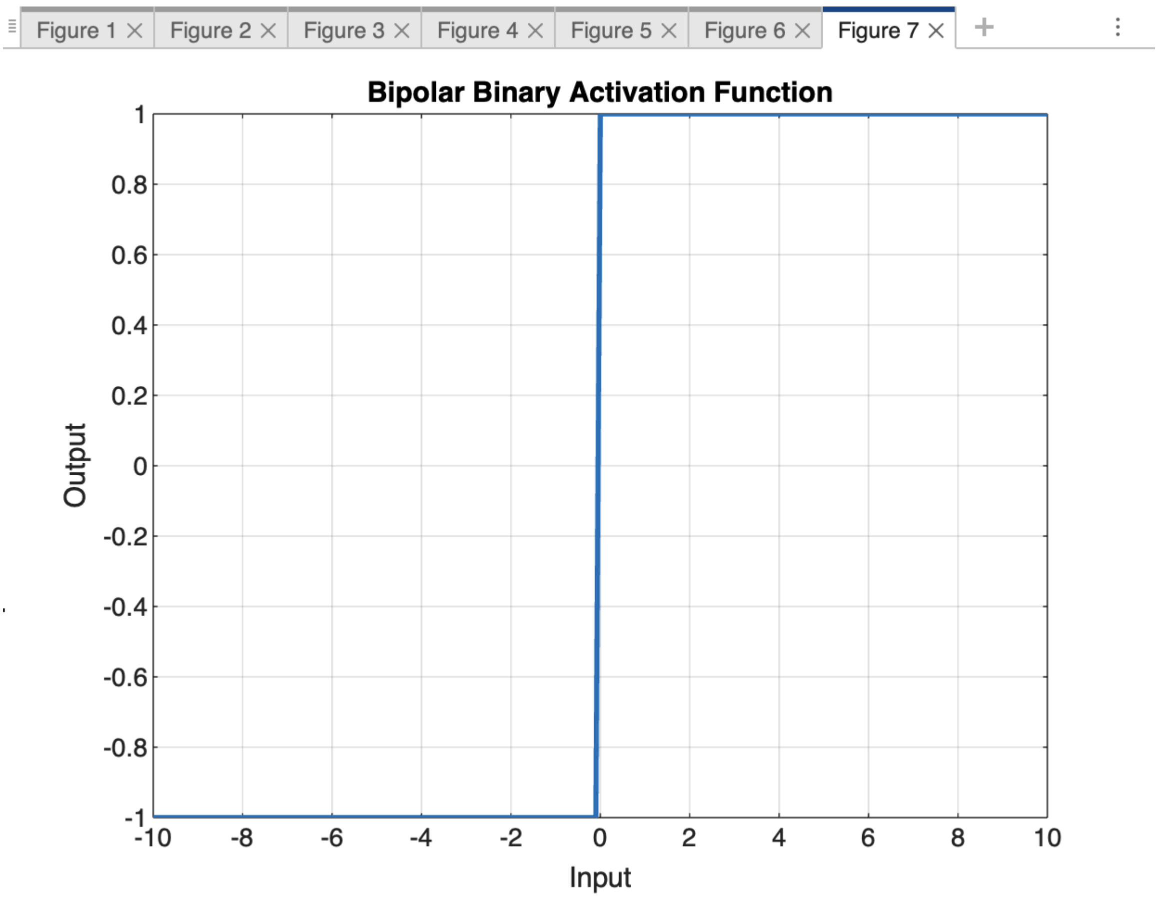


Figure 1
Figure 2
Figure 3
Figure 4
Figure 5
Figure 6
Figure 7
Bipolar Binary Activation Function
Output
Input
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-10
-8
-6
-4
-2
0
2
4
6
8
10

* Conclusion :

Activation functions are crucial for determining neuron output and introducing non-linearity, enabling neural networks to solve complex problems. Understanding them is key to designing effective models.