**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

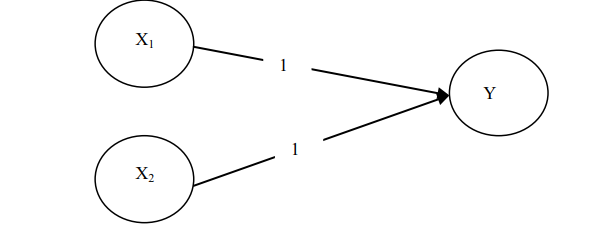
**Learning Outcome:**

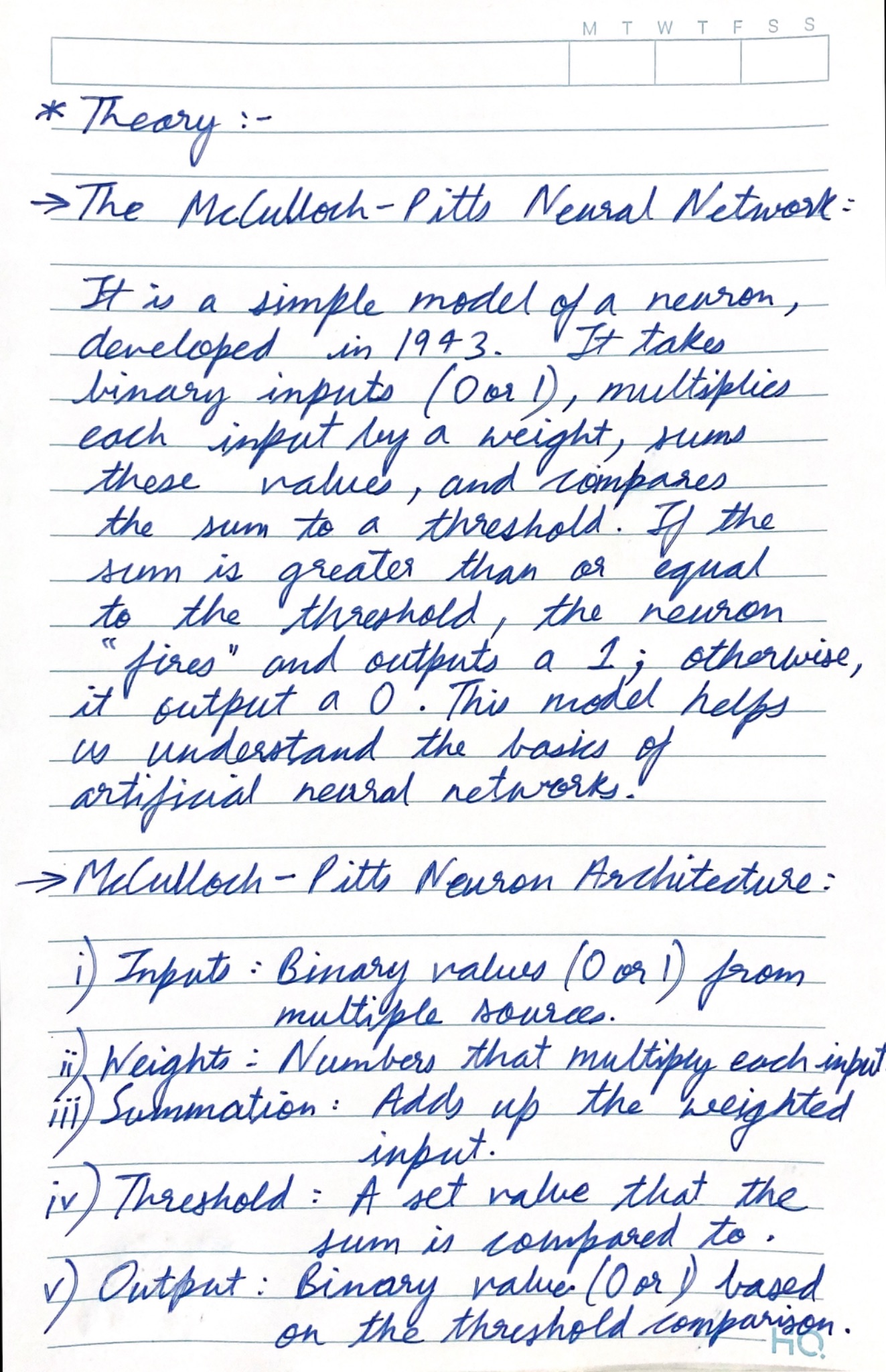
1. To understand the fundamentals of ANN

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

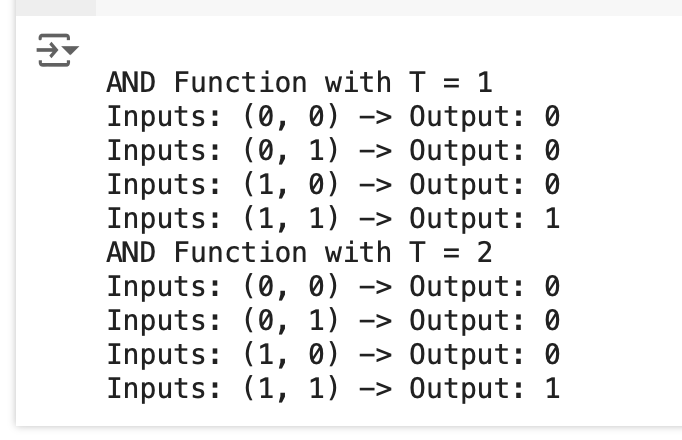


****

**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)}") |
| --- |

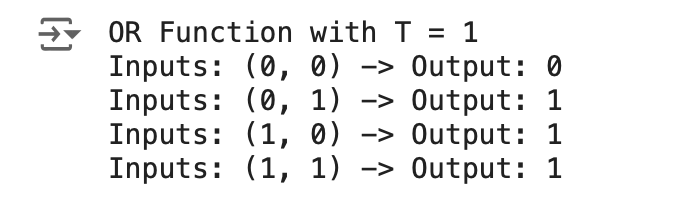
**Output Screenshot:**

****

**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)}") |
| --- |

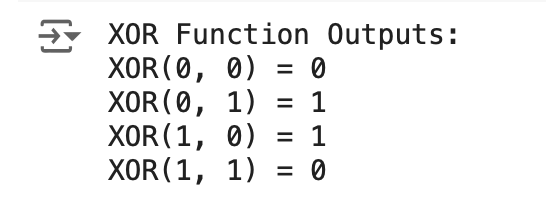
**Output Screenshot:**

****

**Program: c)**

| # 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)}") |
| --- |

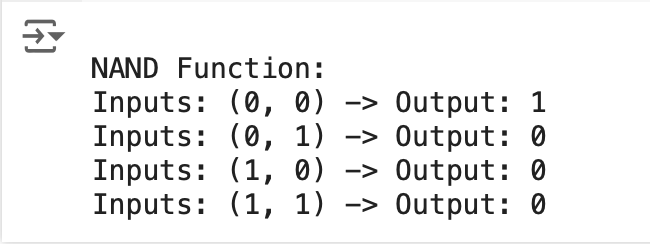
**Output Screenshot:**

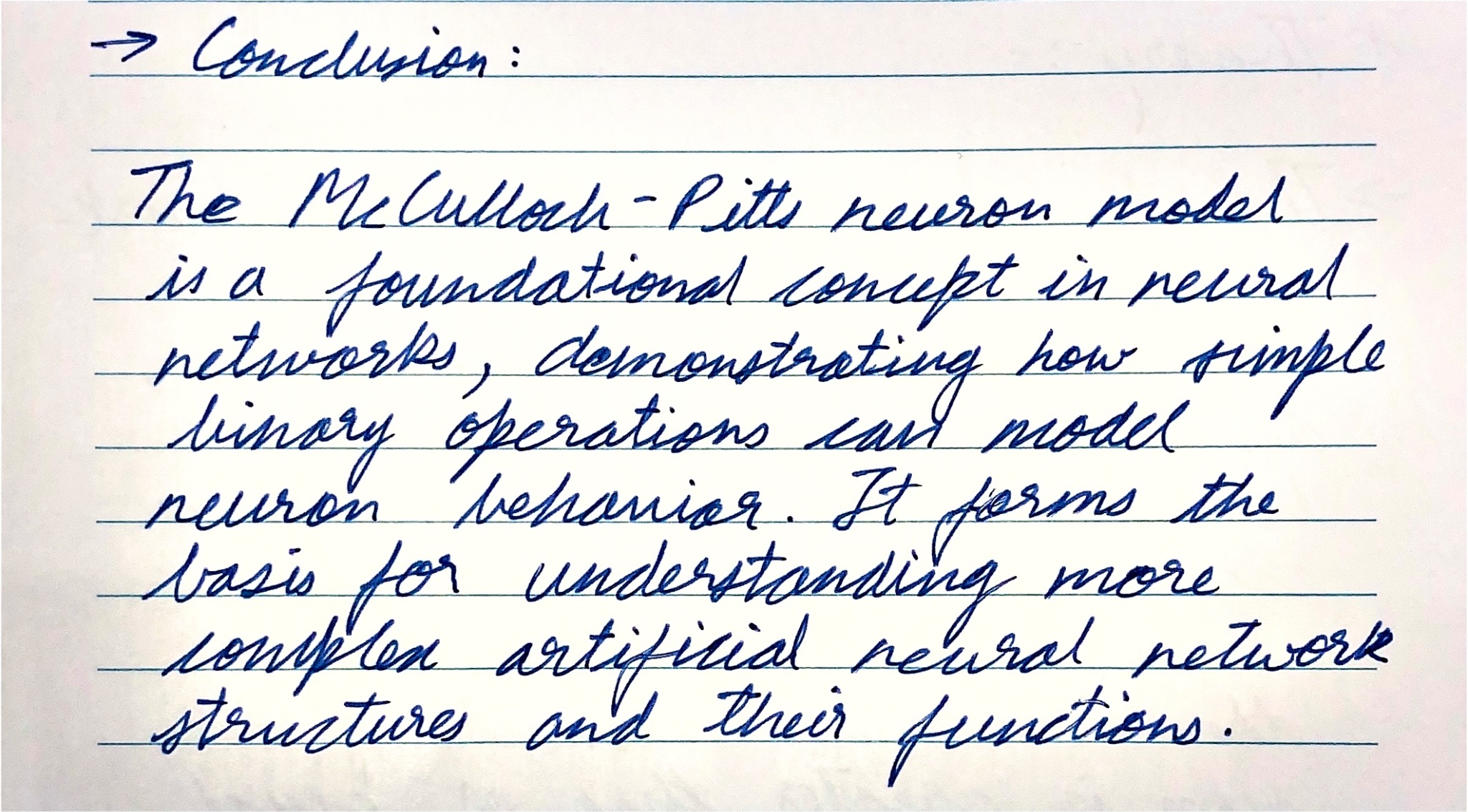
****

**Program d)**

| # 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:**

****

****

**Assignment 1**

**Experiment 3**

**Title: ANN and Its Models**

Name of Student: Sangeet Agrawal PRN No. 21070122140

DoP: 29 Jul DoS: 4 Aug

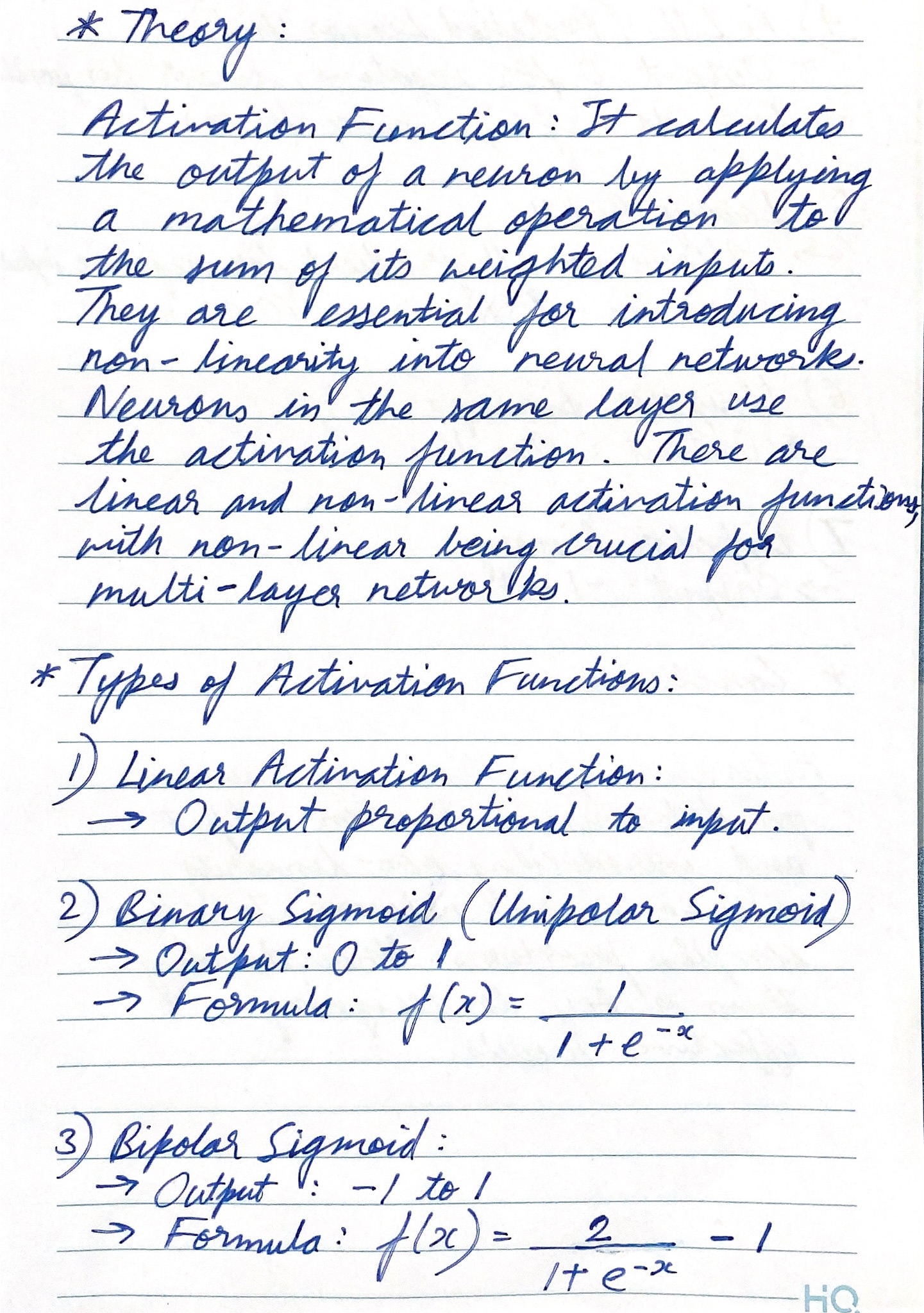
**Aim: e)** Implement activation functions Binary unipolar, binary bipolar, continuous unipolar, sigmoid, and ReLU function

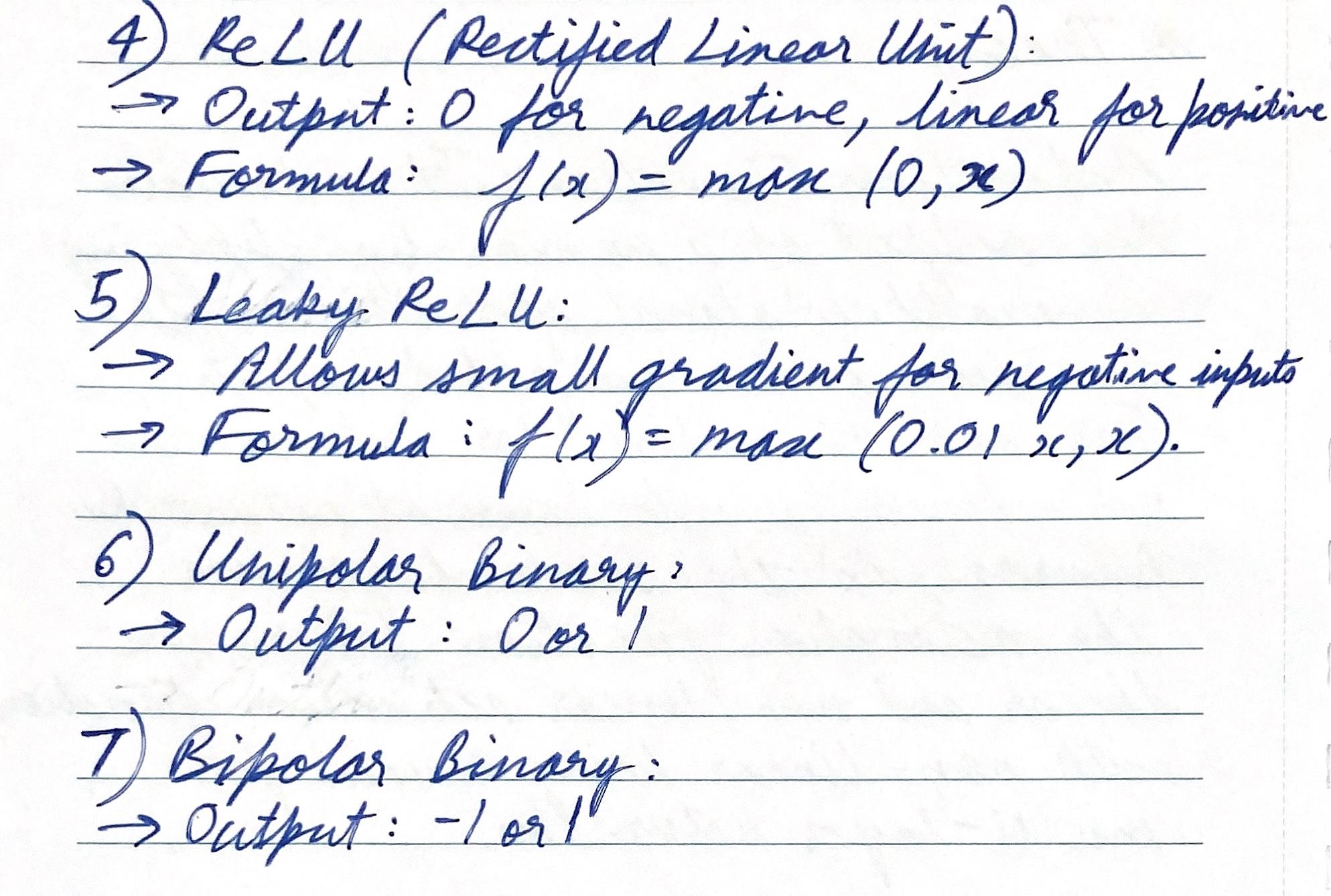
**Learning Outcome:**

1. To understand the fundamentals of ANN

2. To implement McCulloch–Pitts Neuron Model

**Hardware/Software:** MATLAB Online

****

****

**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);

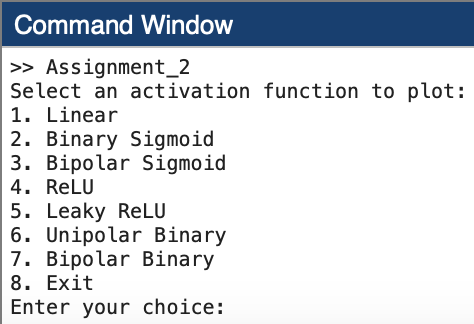
xlabel('Input');

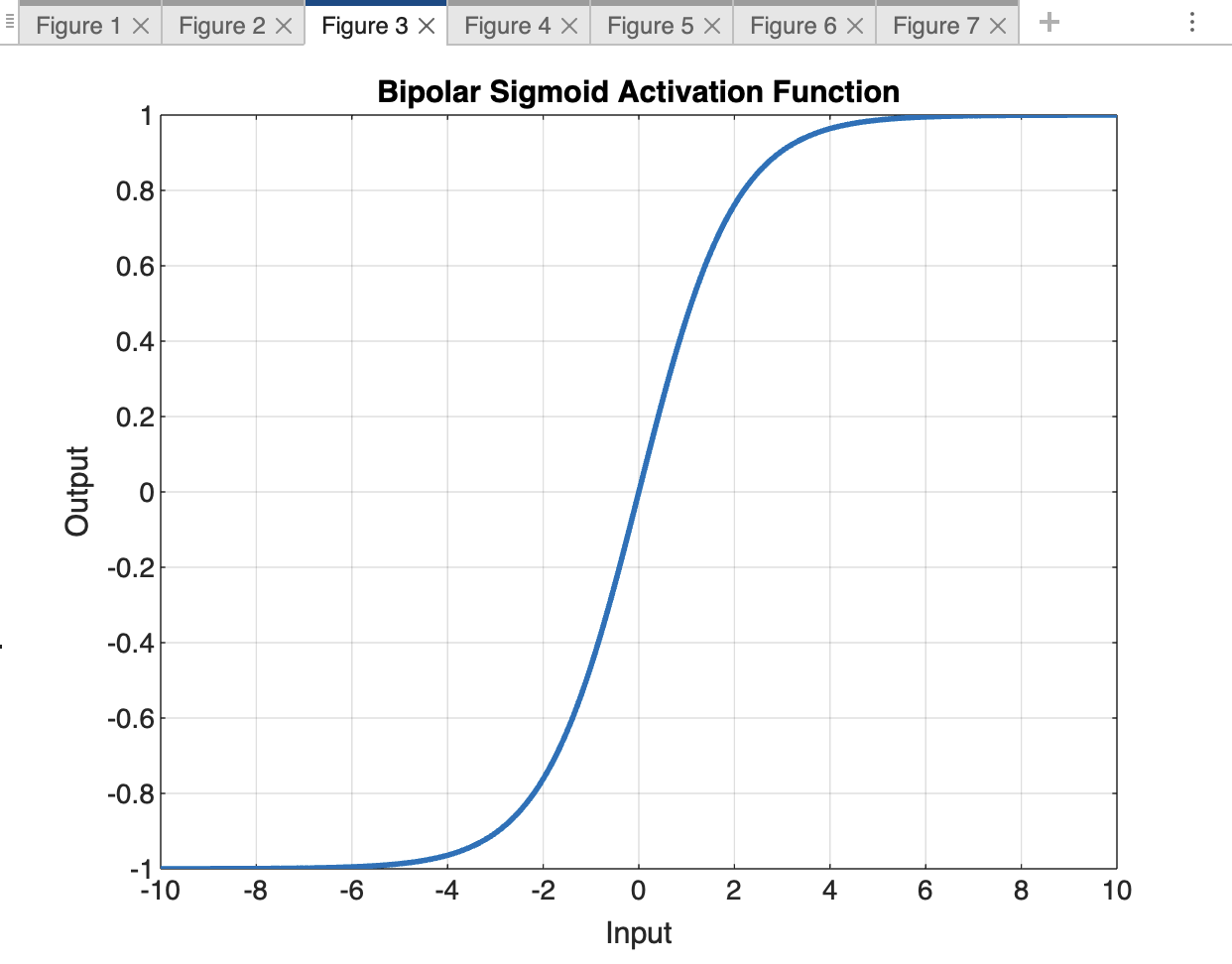
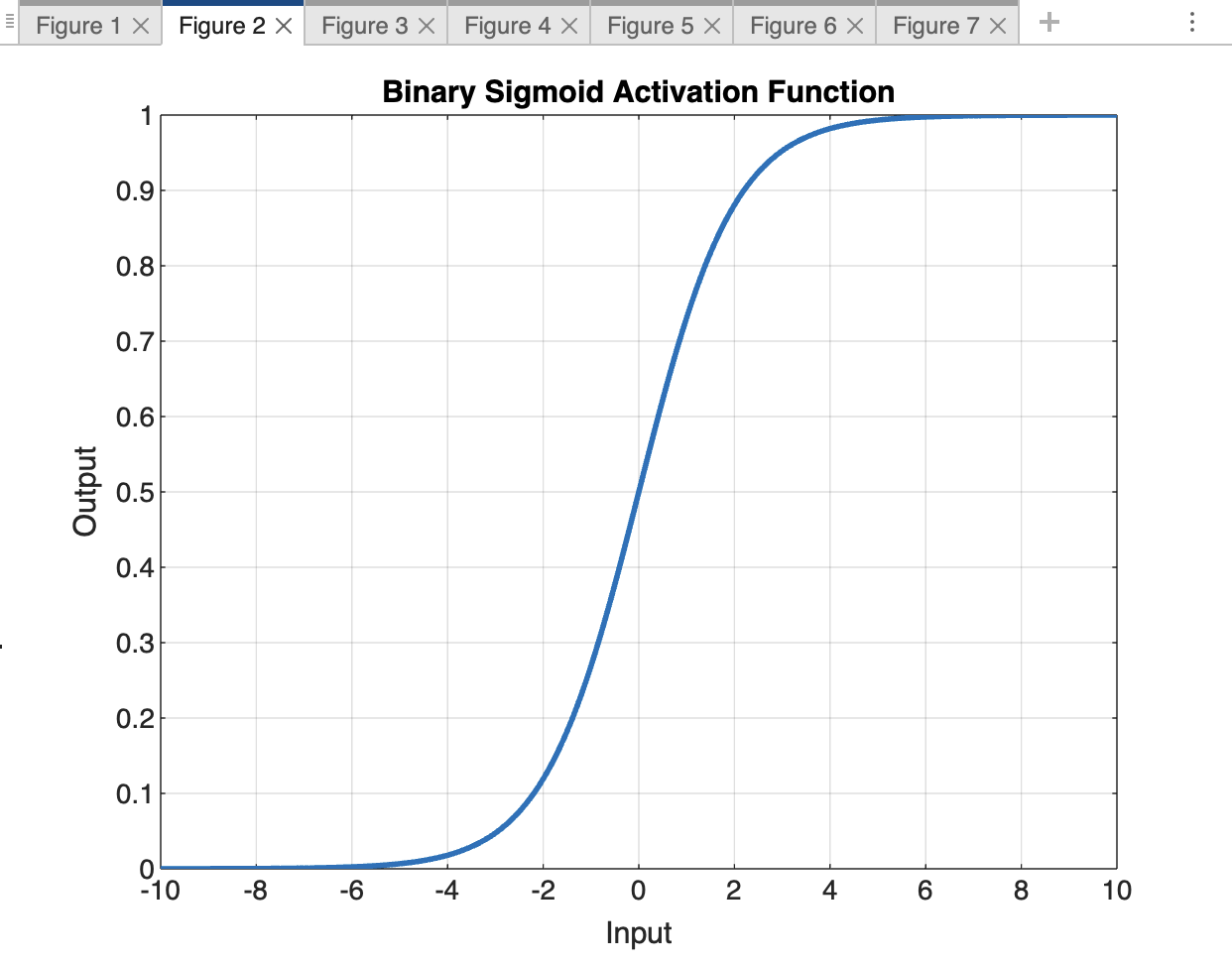
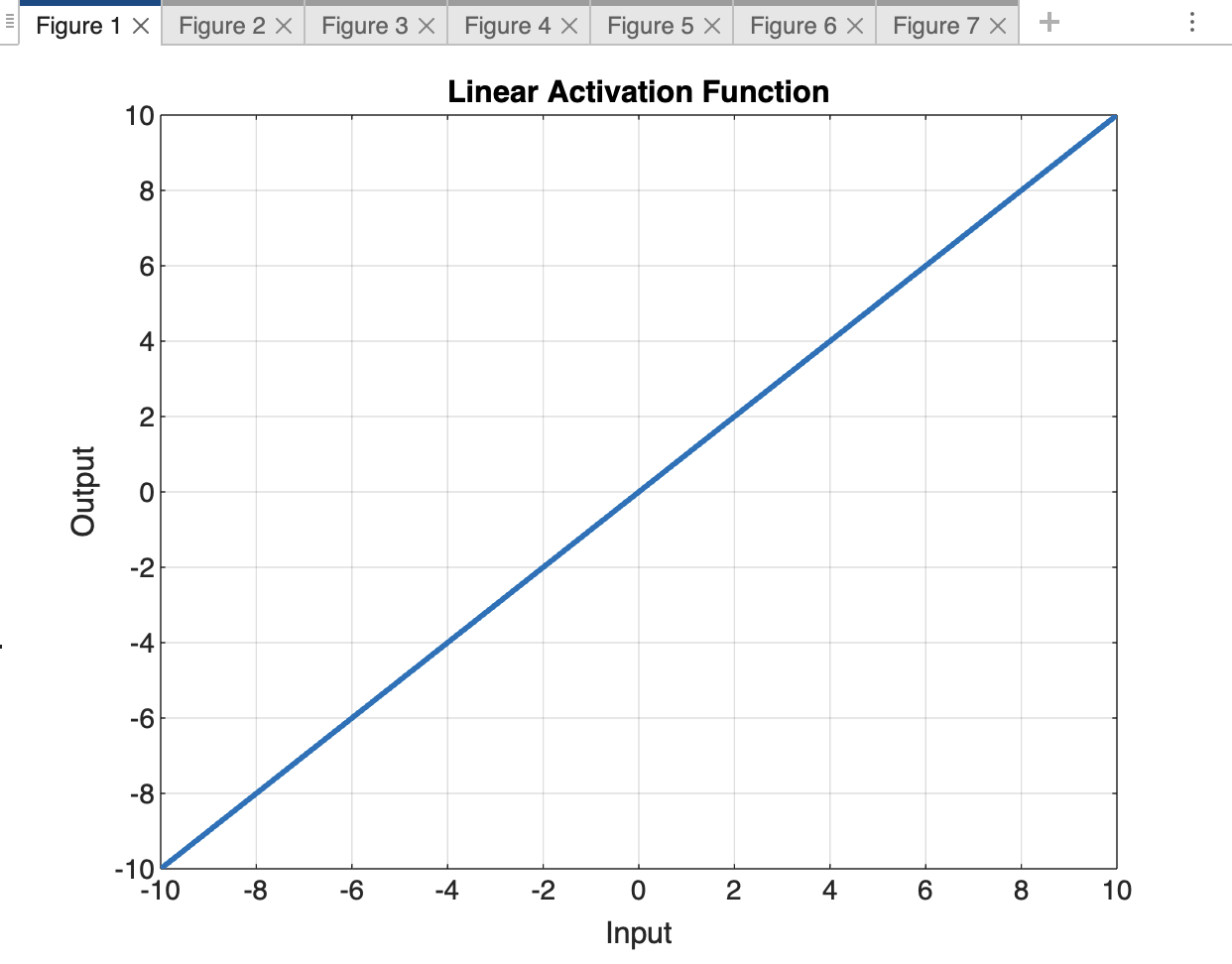
ylabel('Output');

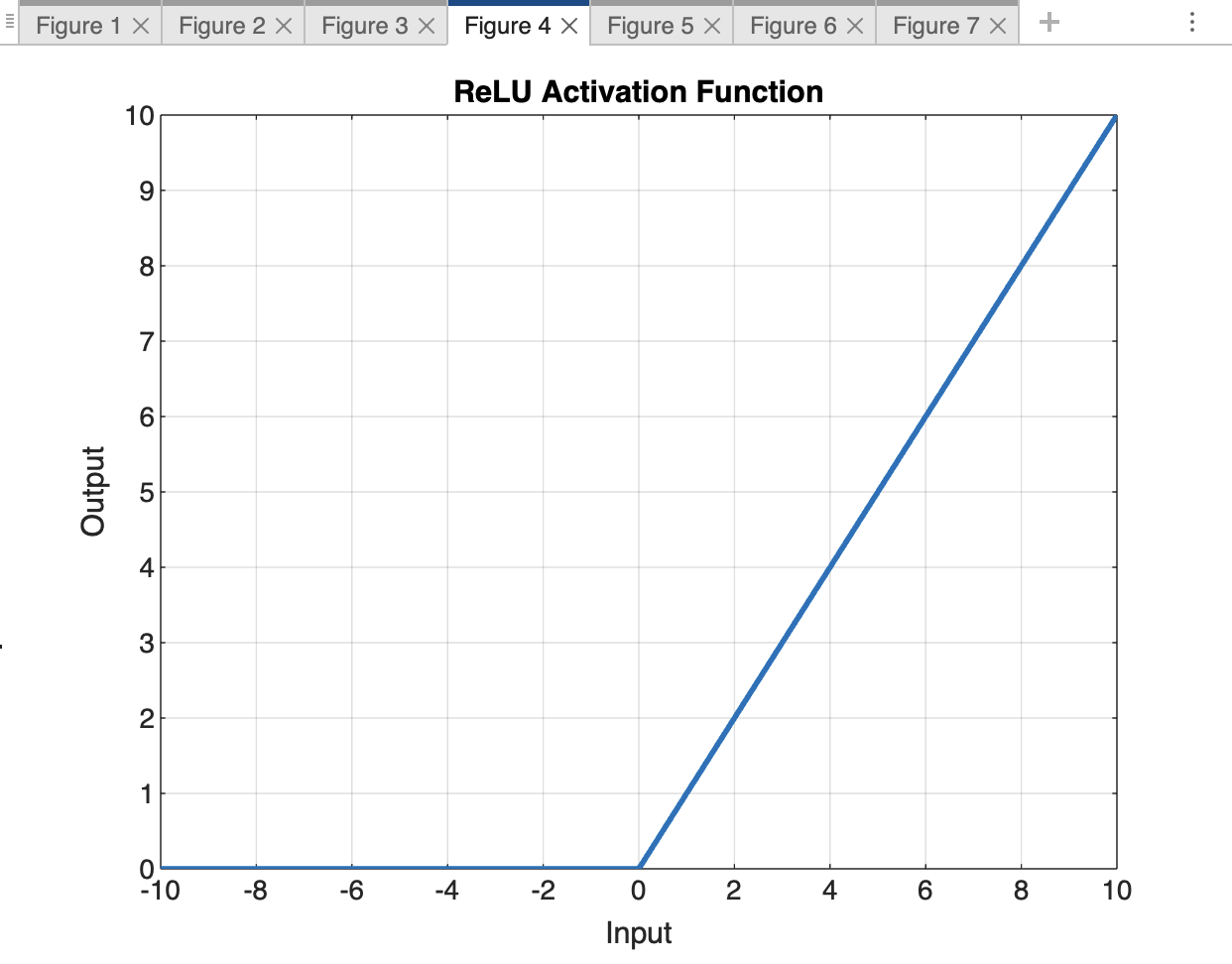
grid on;

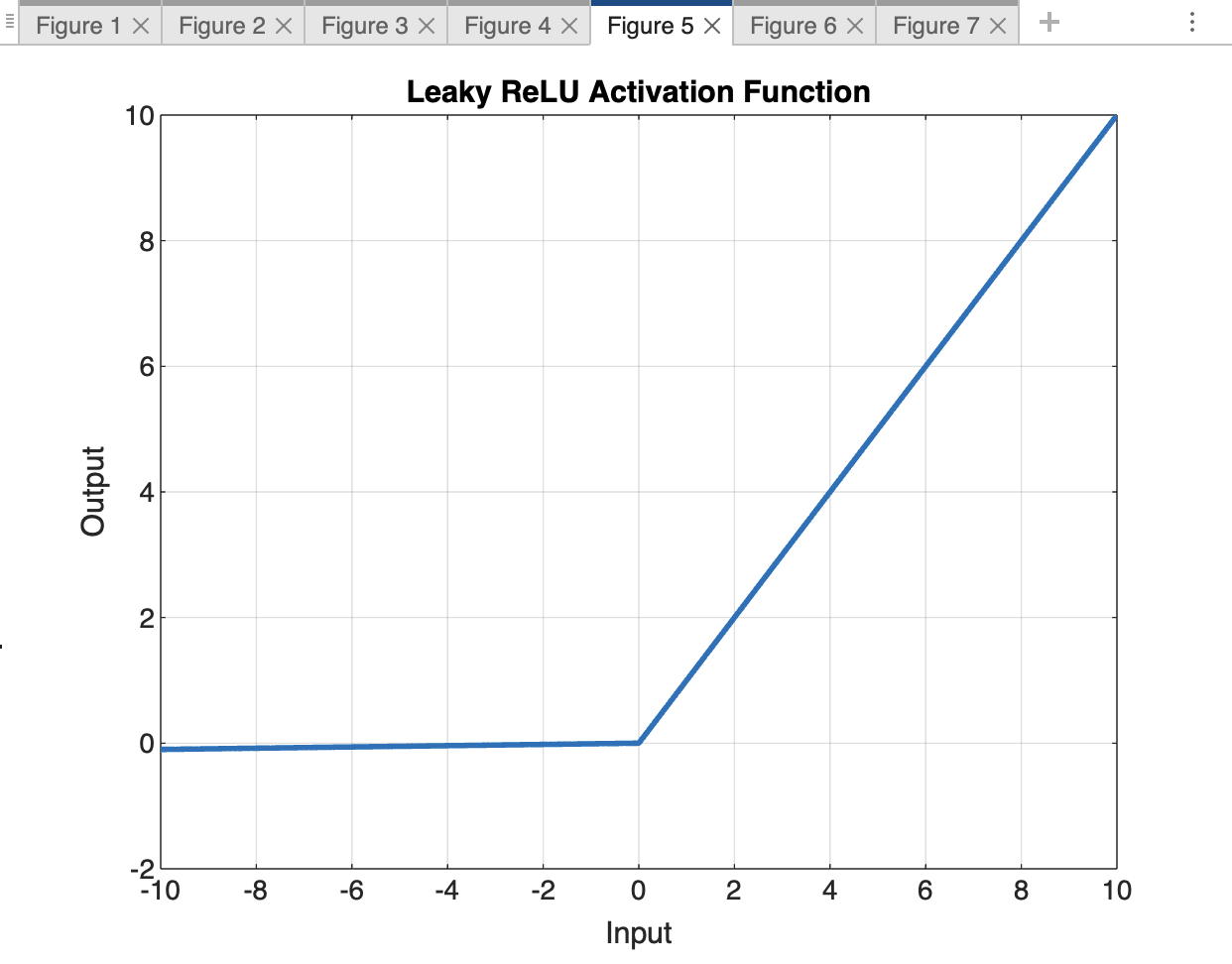
end

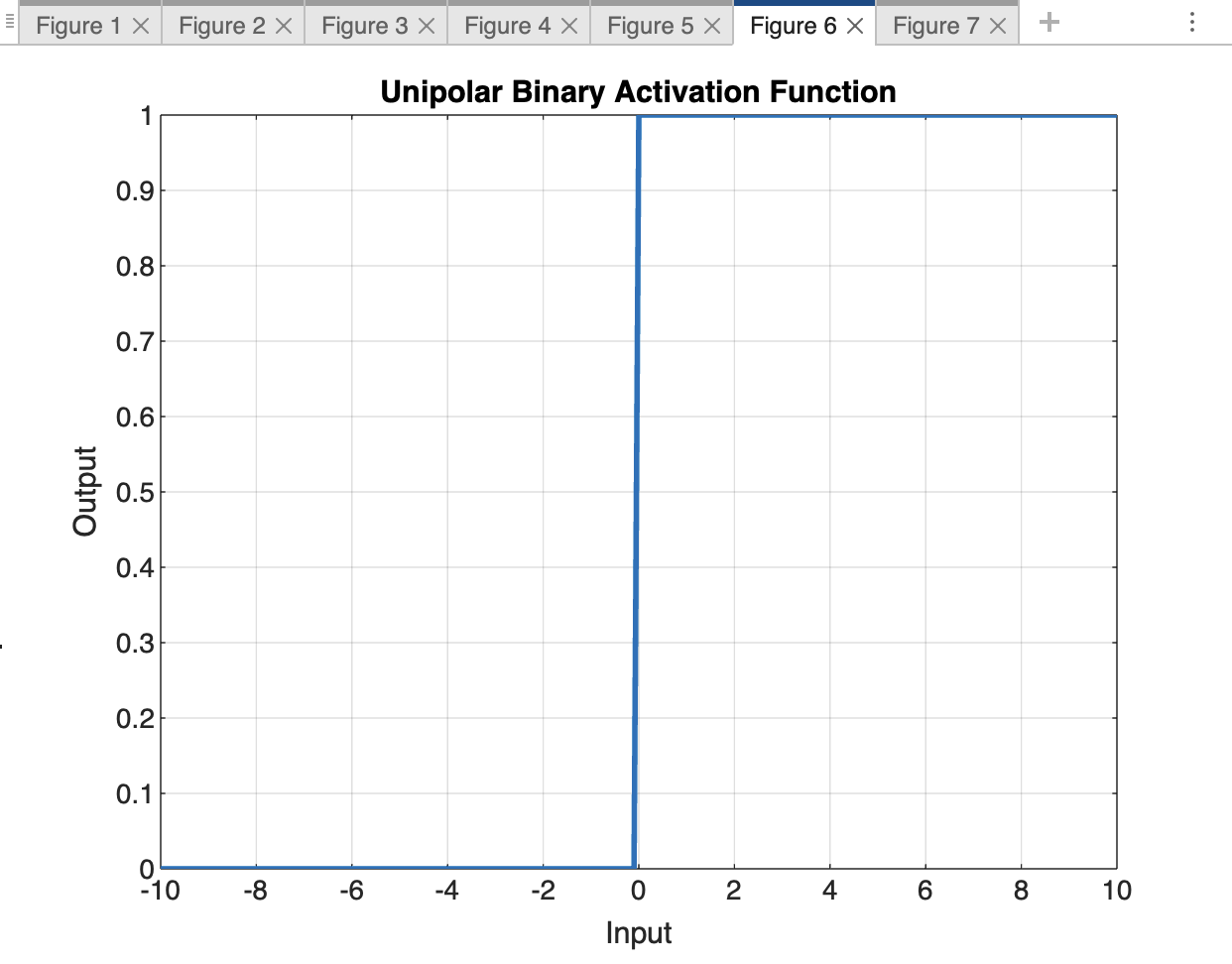
**Output Screenshot:**

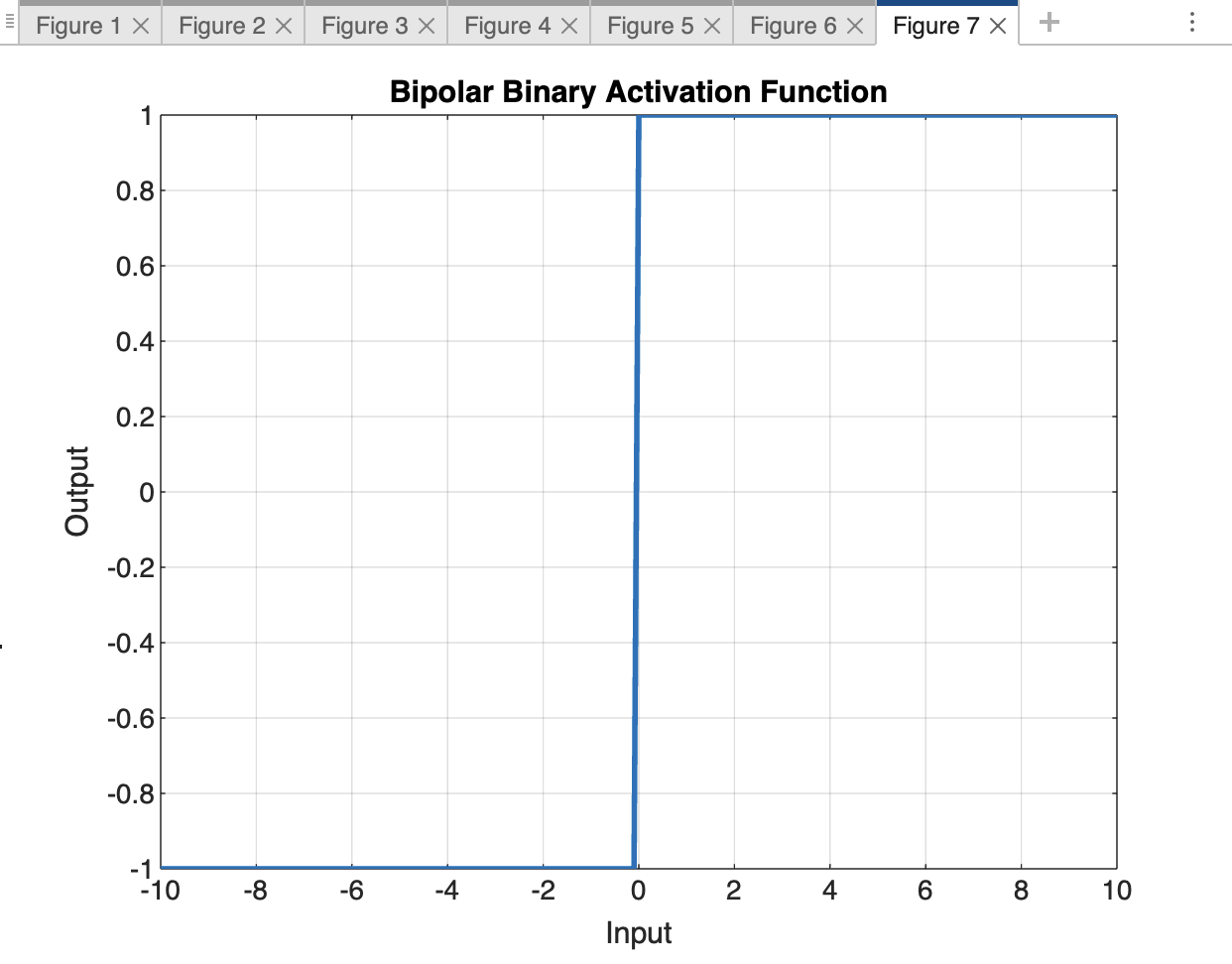


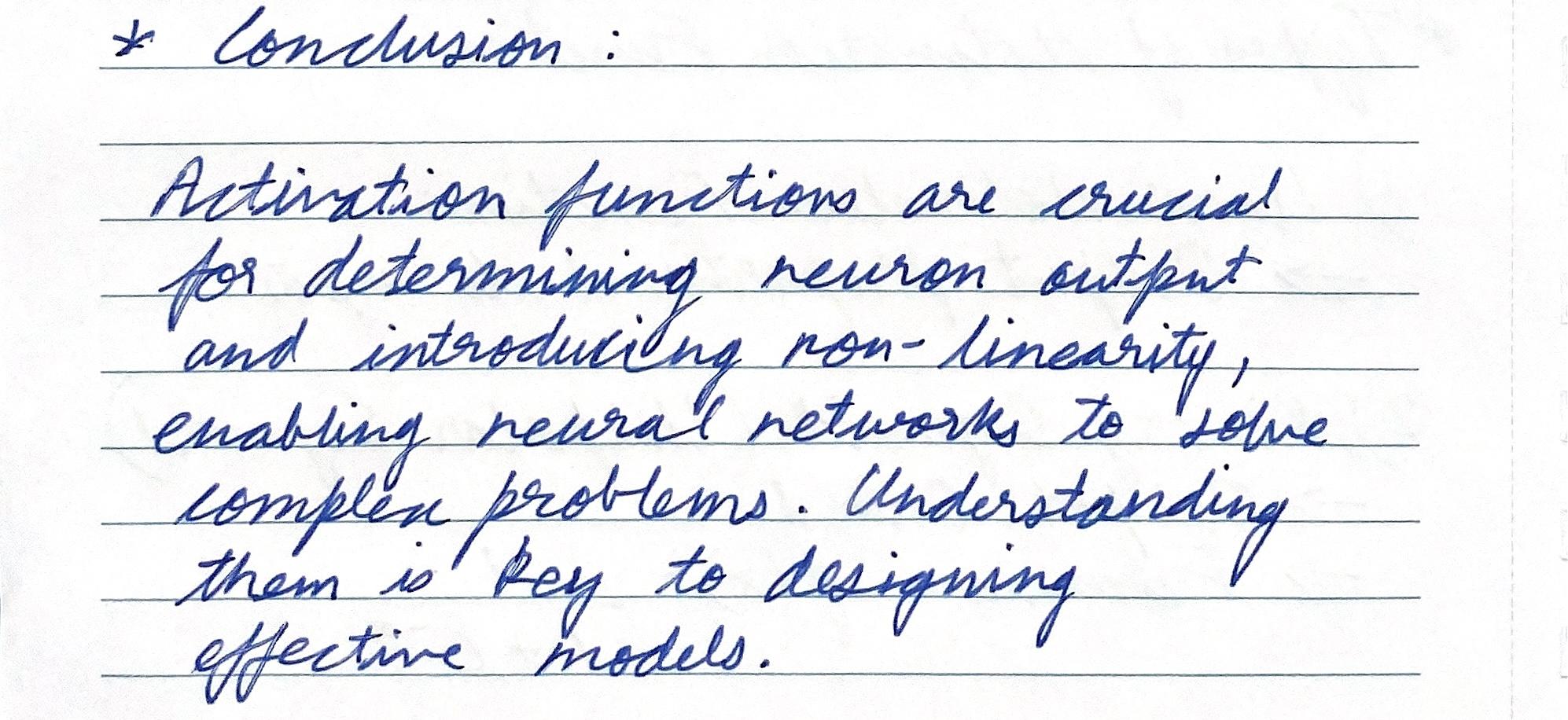
****

****

****

****

****

****