

|  |
| --- |
| **Title: Activation functions.** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Objective:** To implement activation functions of Neural Network.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

CO1 : Identify and describe soft computing techniques and their roles **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

* J.S.R.Jang, C.T.Sun and E.Mizutani, “Neuro-Fuzzy and Soft Computing”, PHI, 2004, Pearson Education 2004.
* Davis E.Goldberg, “Genetic Algorithms: Search, Optimization and Machine Learning”, Addison Wesley, N.Y., 1989.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Pre Lab/ Prior Concepts:**

Neural networks, sometimes referred to as connectionist models, are parallel-distributed models that have several distinguishing features-

1)      A set of processing units;

2)      An activation state for each unit, which is equivalent to the output of the unit;

3)      Connections between the units. Generally each connection is defined by a weight *wjk* that

determines the effect that the signal of unit *j* has on unit *k;*

4)      A propagation rule, which determines the effective input of the unit from its external inputs;

5)      An activation function, which determines the new level of activation based on the effective

input and the current activation;

6)      An external input (bias, offset) for each unit;

7)      A method for information gathering (learning rule);

8)      An environment within which the system can operate, provide input signals and, if necessary, error signals.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Implementation Details:**

Most units in neural network transform their net inputs by using a scalar-to-scalar function called an *activation function*, yielding a value called the unit's activation. Except possibly for output units, the activation value is fed to one or more other units. Activation functions with a bounded range are often called squashing functions. Some of the most commonly used activation functions are :

1. **Identity function**
2. **Binary step function :**
3. **Sigmoid function:**
4. **Bipolar sigmoid function**: this function is defined as follows:
5. **Ramp function**: this function is defined as follows:
6. **ReLU function**: this function is defined as follows

**Code:**

#Implement the 6 Activation functions

import numpy as np

import matplotlib.pyplot as plt

*# Activation functions*

def identity(x):

    return x

def binary\_step(x):

    return np.where(x >= 0, 1, 0)

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def bipolar\_sigmoid(x):

    return (2 / (1 + np.exp(-x))) - 1

def ramp(x):

    return np.maximum(0, x)

*# Input and weight vectors*

input\_vector = np.array(list(map(float, input("Enter the input vector (space-separated): ").split())))

weight\_vector = np.array(list(map(float, input("Enter the weight vector (space-separated): ").split())))

*# Ensure both vectors have the same length*

if len(input\_vector) != len(weight\_vector):

    raise ValueError("Input and weight vectors must have the same length")

*# Calculate y\_n*

y\_n = np.dot(input\_vector, weight\_vector)

*# Range of x values for plotting (centered around y\_n for better visualization)*

x = np.linspace(y\_n - 10, y\_n + 10, 1000)

*# Plot the activation functions*

plt.figure(figsize=(12, 8))

plt.subplot(2, 3, 1)

plt.plot(x, identity(x), label='Identity')

plt.axvline(x=y\_n, color='r', linestyle='--', label=f'y\_n = {y\_n:.2f}')

plt.title('Identity Function')

plt.grid()

plt.legend()

plt.subplot(2, 3, 2)

plt.plot(x, binary\_step(x), label='Binary Step')

plt.axvline(x=y\_n, color='r', linestyle='--', label=f'y\_n = {y\_n:.2f}')

plt.title('Binary Step Function')

plt.grid()

plt.legend()

plt.subplot(2, 3, 3)

plt.plot(x, sigmoid(x), label='Sigmoid')

plt.axvline(x=y\_n, color='r', linestyle='--', label=f'y\_n = {y\_n:.2f}')

plt.title('Sigmoid Function')

plt.grid()

plt.legend()

plt.subplot(2, 3, 4)

plt.plot(x, bipolar\_sigmoid(x), label='Bipolar')

plt.axvline(x=y\_n, color='r', linestyle='--', label=f'y\_n = {y\_n:.2f}')

plt.title('Bipolar Function')

plt.grid()

plt.legend()

plt.subplot(2, 3, 5)

plt.plot(x, ramp(x), label='Ramp')

plt.axvline(x=y\_n, color='r', linestyle='--', label=f'y\_n = {y\_n:.2f}')

plt.title('Ramp Function')

plt.grid()

plt.legend()

plt.tight\_layout()

plt.show()

*# Display the calculated y\_n and activation values*

print(f"y\_n (weighted sum): {y\_n}")

print(f"Identity: {identity(y\_n)}")

print(f"Binary Step: {binary\_step(y\_n)}")

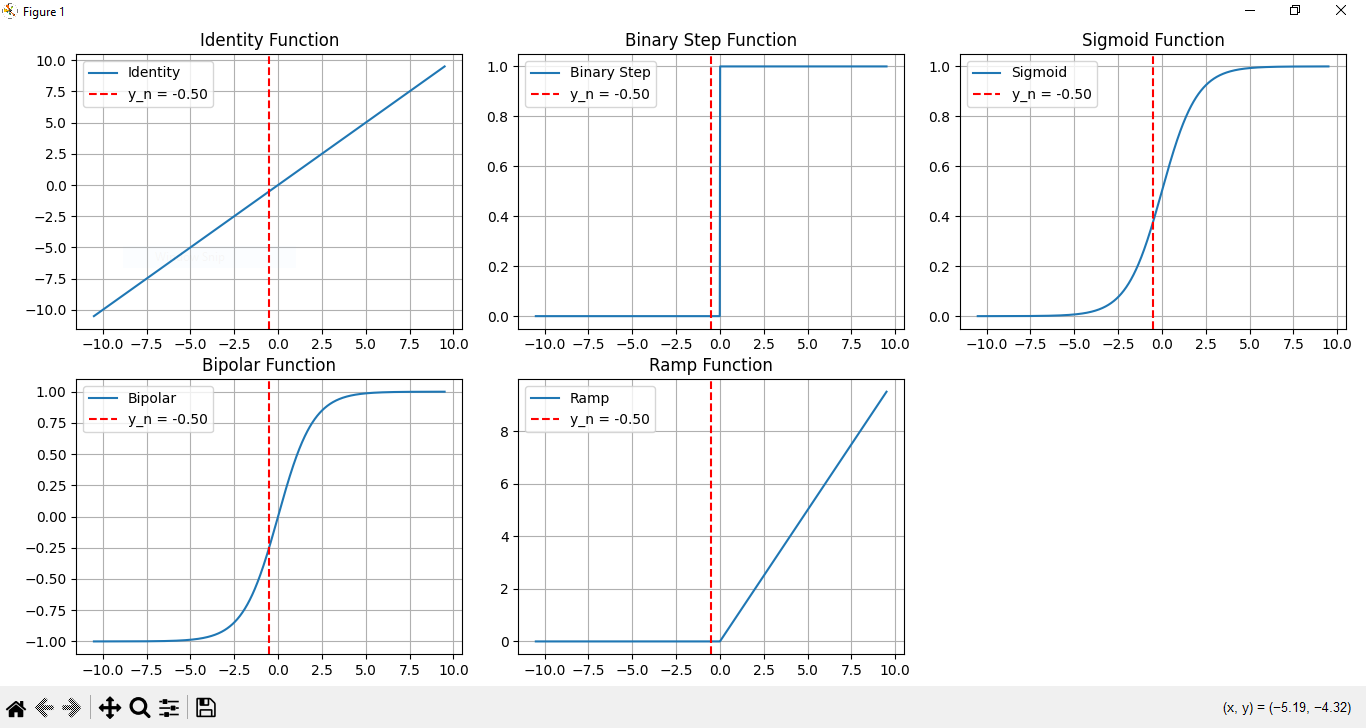
print(f"Sigmoid: {sigmoid(y\_n)}")

print(f"Bipolar: {bipolar\_sigmoid(y\_n)}")

print(f"Ramp: {ramp(y\_n)}")

**Output:**

#Plot the graph of the output of each activation function.



**Conclusion:** Thus, we have successfully implemented 6 Activation Functions of Neural Network.

**Post Lab Descriptive Questions :**

1. Explain the concept behind using Activation function.

**Ans:**

**Activation functions introduce non-linearity into neural networks, enabling them to learn complex patterns. They determine whether a neuron should be activated and contribute to the decision-making process. Without activation functions, networks would be limited to linear mappings, reducing their ability to model complex data.**

1. Explain the different properties of activation functions.

**Ans:**

**Non-linearity**: Allows the network to learn complex relationships. Examples: ReLU, Sigmoid, Tanh.

**Range**: The output range, affecting saturation and gradient flow. Examples:

* Sigmoid: (0,1)(0, 1)(0,1)
* Tanh: (−1,1)(-1, 1)(−1,1)
* ReLU: [0,∞)[0, \infty)[0,∞)

**Differentiability**: Essential for backpropagation and gradient-based optimization. Most activation functions, like Sigmoid and Tanh, are differentiable.

**Saturation**: Regions where gradients are very small, slowing down learning. Notable in Sigmoid and Tanh.

**Computational Efficiency**: Affects training speed and scalability. ReLU is efficient due to its simple thresholding.

**Monotonicity**: Ensures the gradient does not change sign, aiding optimization. Sigmoid is an example of a monotonic function.

**Continuity**: Smooth output transitions, important for optimization. Most standard activation functions are continuous.

.

**Date: 30/06/2024 Signature of faculty in-charge**