Batch: 1 **Roll No.: 1911027**

Experiment No. 01

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

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.
- S. Rajasekaran and G.A.V.Pai, "Neural Networks, Fuzzy Logic and Genetic Algorithms", PHI, 2003.
- http://library.thinkquest.org/C007395/tqweb/history.html

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 w_{jk} 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:** In mathematics, an identity function, also called an identity relation or identity map or identity transformation, is a function that always returns the same value that was used as its argument. This function is defined as f(x) = x for all x. So, the function definition implies that output is same as the input.

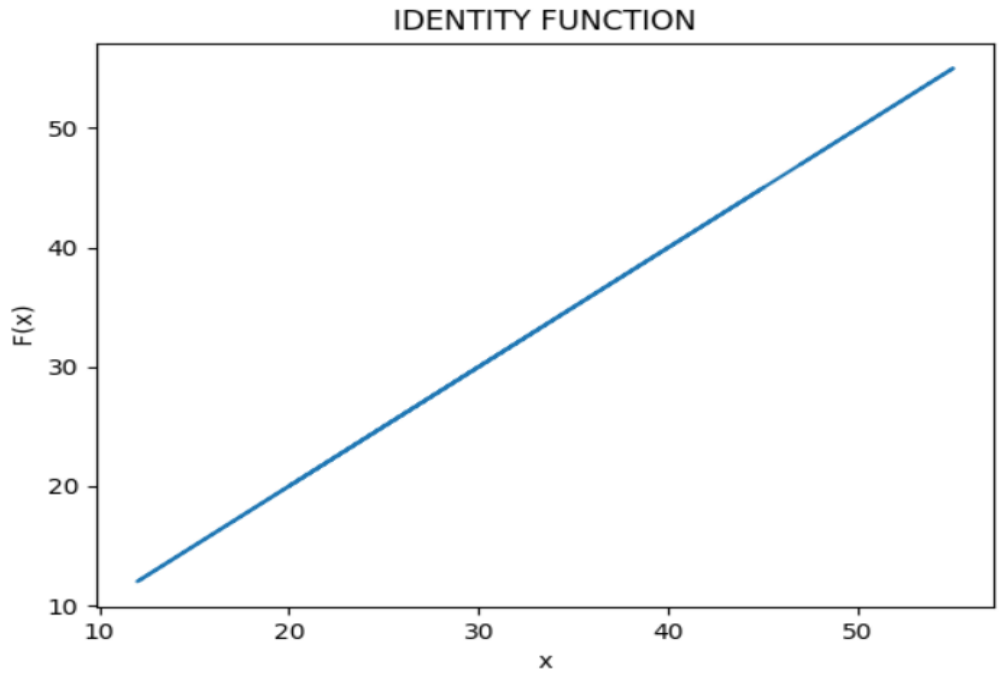
```
import matplotlib.pyplot as plt
print("-----------------------------------------------------------------")
print("\t\t\tIDENTITY FUNCTION")
print("-----------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("-----------------------------------------------------------------")
x=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n-----------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    print(i," ",end="")
print("\n-----------------------------------------------------------------")
f = x
```

```
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('IDENTITY FUNCTION')
plt.show()
```

OUTPUT:

```
------------------------------------------------------------
                    IDENTITY FUNCTION
------------------------------------------------------------
Enter input value X : 33 12 45 20 55 47
------------------------------------------------------------
INPUT (X) : 33  12  45  20  55  47
------------------------------------------------------------
OUTPUT : 33  12  45  20  55  47
------------------------------------------------------------
```

PLOT:



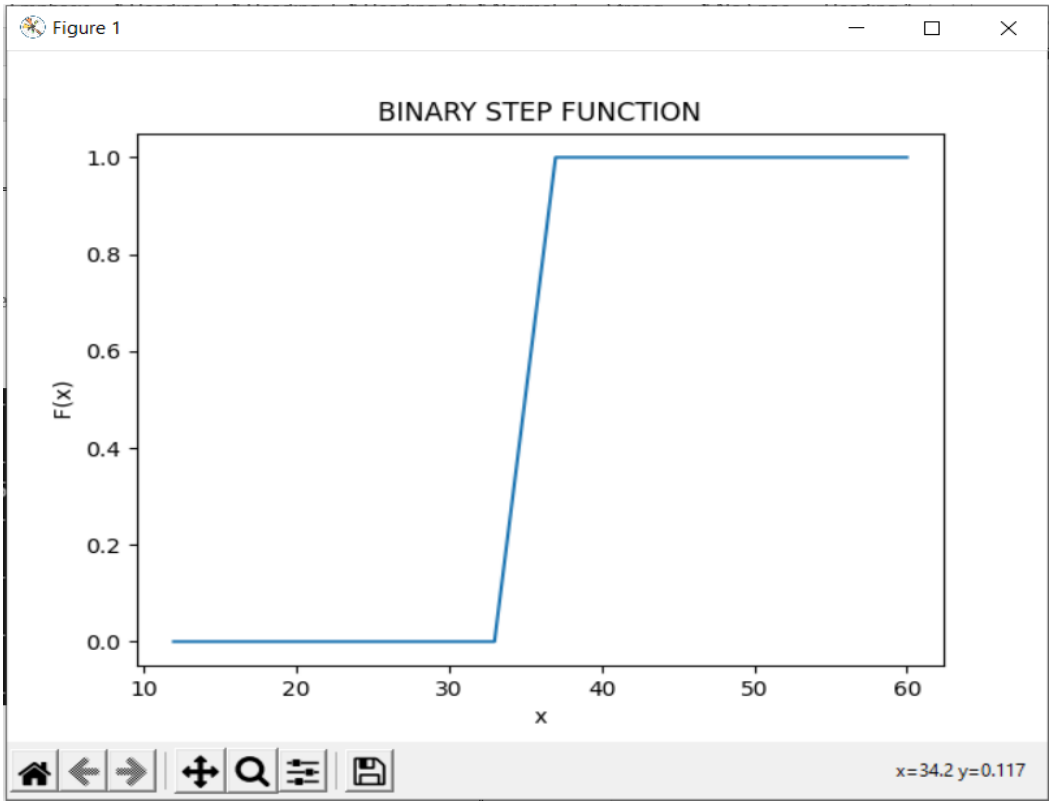
2) **Binary step function:** A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer. Here, f(x) = 1 if x > threshold, f(x) = 0 if x < threshold.

```
import matplotlib.pyplot as plt
print("-----------------------------------------------------------")
print("\t\t\tBINARY STEP FUNCTION")
print("-----------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("-----------------------------------------------------------")
theta=int(input("Enter threshold value (Theta) : "))
print("-----------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n-----------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    if(i>=theta):
        f.append(1)
        print("1 ",end="")
    else:
        f.append(0)
        print("0 ",end="")
print("\n-----------------------------------------------------------")
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('BINARY STEP FUNCTION')
plt.show()
```

OUTPUT:

```
------------------------------------------------------------
                    BINARY STEP FUNCTION
------------------------------------------------------------
Enter input value X : 12 25 33 37 45 60
------------------------------------------------------------
Enter threshold value (Theta) : 35
------------------------------------------------------------
INPUT (X) : 12  25  33  37  45  60
------------------------------------------------------------
OUTPUT : 0 0 0 1 1 1
------------------------------------------------------------
```

PLOT:



3) **Bipolar step function:** This function can be defined as,

$$f(x)=\begin{cases}1, & \text{if } x \geq \theta \\ -1, & \text{if } x < \theta\end{cases}$$

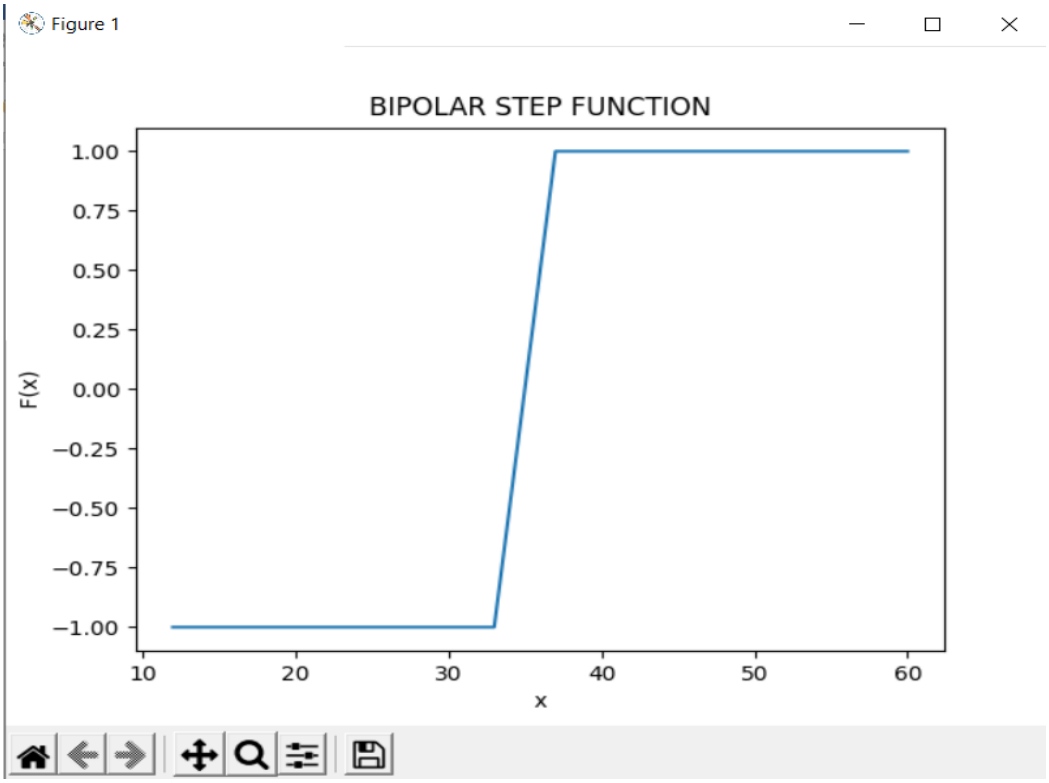
where θ represents the threshold value. This function is also used in a single layer net to convert the net input to an output that is bipolar (+1 or -1).

```
import matplotlib.pyplot as plt
print("------------------------------------------------------------")
print("\t\t\tBIPOLAR STEP FUNCTION")
print("------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("------------------------------------------------------------")
theta=int(input("Enter threshold value (Theta) : "))
print("------------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    if(i>=theta):
        f.append(1)
        print("1 ",end="")
    else:
        f.append(-1)
        print("-1 ",end="")
print("\n------------------------------------------------------------")
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('BIPOLAR STEP FUNCTION')
plt.show()
```

OUTPUT:

```
------------------------------------------------------------------
                      BIPOLAR STEP FUNCTION
------------------------------------------------------------------
Enter input value X : 12 25 33 37 45 60
------------------------------------------------------------------
Enter threshold value (Theta) : 35
------------------------------------------------------------------
INPUT (X) : 12  25  33  37  45  60
------------------------------------------------------------------
OUTPUT : -1 -1 -1 1 1 1
------------------------------------------------------------------
```

PLOT:



4) **Sigmoid function:** The function the sigmoid functions are widely used in back propagation nets because of the relationship between the value of the functions at a point and the value of the derivative at that point which reduces the computational burden during training.

A) **Binary sigmoid function:** Also called as the logistics sigmoid function or unipolar sigmoid function. It can be defined as,

$$y = f(y_{in})$$

$$f(x) = \frac{1}{1+e^{-\lambda x}} = \frac{1}{1+e^{-y_{in}}}$$

where λ is the steepness parameter.

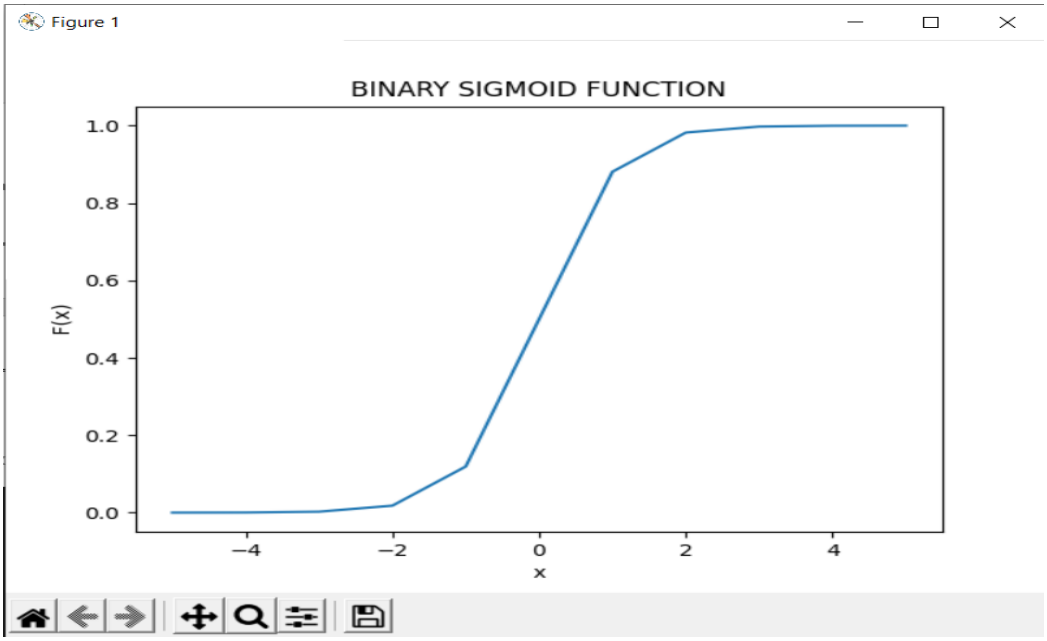
```
import matplotlib.pyplot as plt
import math
print("----------------------------------------------------------------")
print("\t\t\tBINARY SIGMOID FUNCTION")
print("----------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("----------------------------------------------------------------")
lambd=int(input("Enter steepness parameter (Lambda) : "))
print("----------------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n----------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    t=1/(1+math.exp(-1*lambd*i))
    f.append(t)
    print(t," ",end="")
print("\n----------------------------------------------------------------")
plt.plot(x, f)
```

```
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('BINARY SIGMOID FUNCTION')
plt.show()
```

OUTPUT:

```
------------------------------------------------------------
                    BINARY SIGMOID FUNCTION
------------------------------------------------------------
Enter input value X : -5 -4 -3 -2 -1 0 1 2 3 4 5
------------------------------------------------------------
Enter steepness parameter (Lambda) : 2
------------------------------------------------------------
INPUT (X) : -5  -4  -3  -2  -1  0  1  2  3  4  5
------------------------------------------------------------
OUTPUT : 4.5397868702434395e-05  0.0003353501304664781  0.002472623156634743  0.01798620996209156  0.11920292202211755  0.5  0.8807970779778823  0.9820137900379085  0.9975273768
8433653  0.9996646498695336  0.9999546021312976
------------------------------------------------------------
```

PLOT:



B) Unipolar sigmoid function: This function is given by

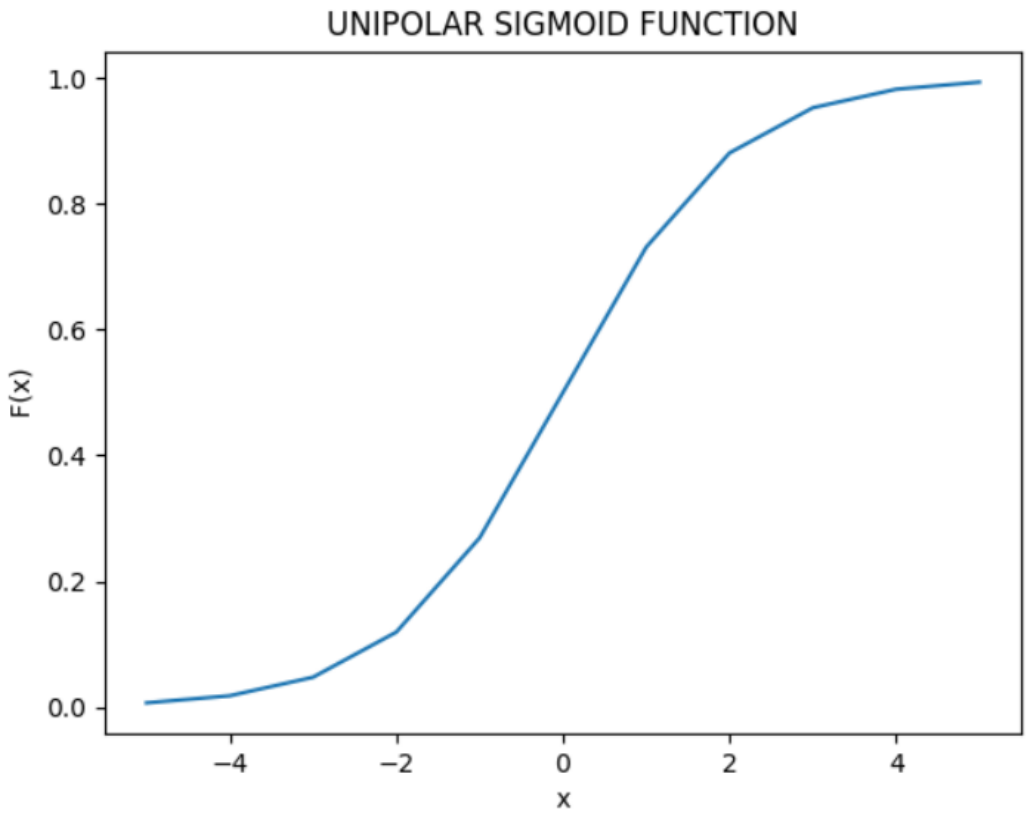
$$f(x) = \frac{1}{1 + e^{-x}}$$

```
import matplotlib.pyplot as plt
import math
print("-----------------------------------------------------------------")
print("\t\t\tUNIPOLAR SIGMOID FUNCTION")
print("-----------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("-----------------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n-----------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    t=1/(1+math.exp(-1*i))
    f.append(t)
    print(t," ",end="")
print("\n-----------------------------------------------------------------")
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('UNIPOLAR SIGMOID FUNCTION')
plt.show()
```

OUTPUT:

```
------------------------------------------------------------
                    UNIPOLAR SIGMOID FUNCTION
------------------------------------------------------------
Enter input value X : -5 -4 -3 -2 -1 0 1 2 3 4 5
------------------------------------------------------------
INPUT (X) : -5  -4  -3  -2  -1  0  1  2  3  4  5
------------------------------------------------------------
OUTPUT : 0.006692850924284855 4 0.01798620996209156  0.04742587317756678  0.11920292202211755  0.2689414213699951  0.5  0.7310585786300049  0.8807970779778823  0.9525741268224334  0.9820137900379085  0.9933071490757153
------------------------------------------------------------
```

PLOT:



C) **Bipolar sigmoid function:** The bipolar sigmoid function is closely related to, hyperbolic tangent function If the network uses the binary data, then it is better to convert it to bipolar form and use the bipolar sigmoidal activation function or hyperbolic tangent function. This function can be defined as,

$$y = f(y_{in})$$

$$f(x) = \frac{2}{1+e^{-\lambda x}} - 1 = \frac{1-e^{-\lambda x}}{1+e^{-\lambda x}}$$

$$f(y_{in}) = \frac{1-e^{-y_{in}}}{1+e^{-y_{in}}}$$

where λ is equal to steepness parameter.

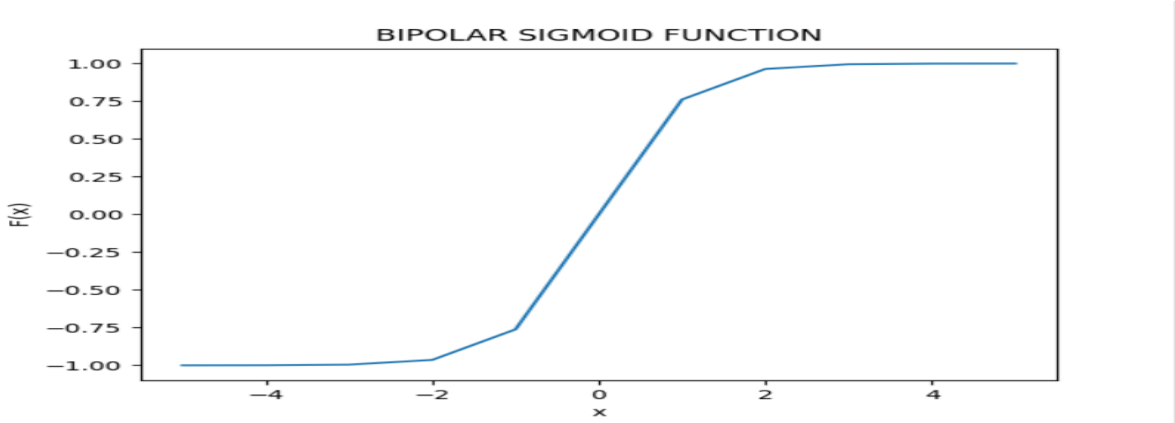
```
import matplotlib.pyplot as plt
import math
print("------------------------------------------------------------")
print("\t\t\tBIPOLAR SIGMOID FUNCTION")
print("------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("------------------------------------------------------------")
lambd=int(input("Enter steepness parameter (Lambda) : "))
print("------------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    t=(1-math.exp(-1*lambd*i))/(1+math.exp(-1*lambd*i))
    f.append(t)
    print(t," ",end="")
```

```
print("\n----------------------------------------------------------------")
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('BIPOLAR SIGMOID FUNCTION')
plt.show()
```

OUTPUT:

```
----------------------------------------------------------------
                    BIPOLAR SIGMOID FUNCTION
----------------------------------------------------------------
Enter input value X : -5 -4 -3 -2 -1 1 2 3 4 5
----------------------------------------------------------------
Enter steepness parameter (Lambda) : 2
----------------------------------------------------------------
INPUT (X) : -5  -4  -3  -2  -1  1  2  3  4  5
----------------------------------------------------------------
OUTPUT : -0.9999092042625951  -0.999329299739067  -0.9950547536867305  -0.9640275800758169  -0.7615941559557649  0.7615941559557649  0.9640275800758168  0.9950547536867306  0.99
93292997390671  0.9999092042625951
----------------------------------------------------------------
```

PLOT:



5) Ramp function: The ramp function is defined as,

$$f(x) = \begin{cases} 1, & if \quad x \geq 1 \\ x, & if \quad 0 \leq x \leq 1 \\ 0, & if \quad x < 0 \end{cases}$$

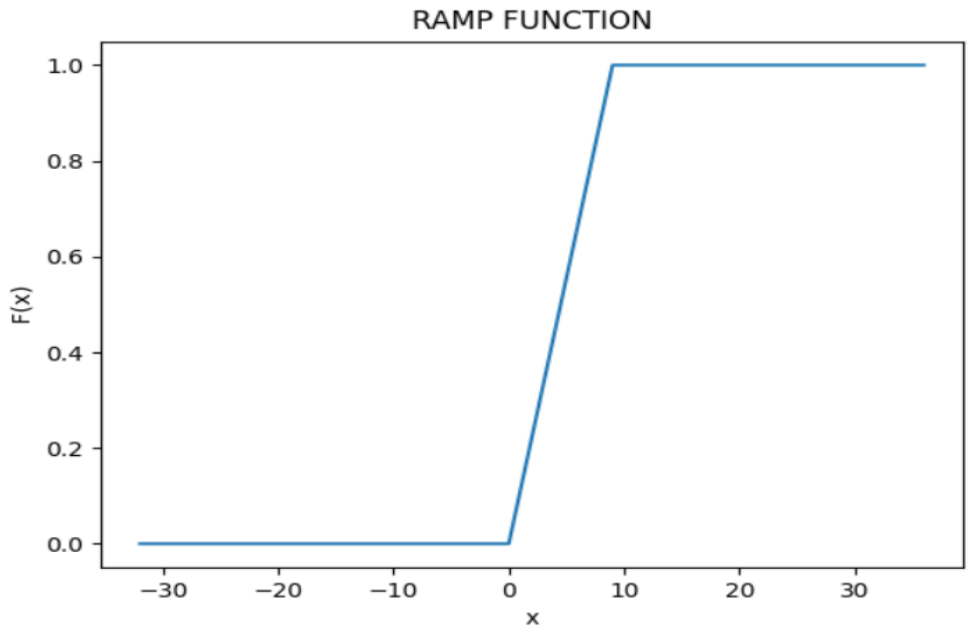
```
import matplotlib.pyplot as plt
import math
print("-----------------------------------------------------------------")
print("\t\t\tRAMP FUNCTION")
print("-----------------------------------------------------------------")
temp=input("Enter input value X : ").split()
print("-----------------------------------------------------------------")
x=[]
f=[]
for i in temp:
    x.append(int(i))
print("INPUT (X) : ",end="")
for i in x:
    print(i," ",end="")
print("\n-----------------------------------------------------------------")
print("OUTPUT : ",end="")
for i in x:
    if(i>1):
        t=1
    elif(i>=0 and i<=1):
        t=i
    else:
        t=0
    f.append(t)
    print(t," ",end="")
print("\n-----------------------------------------------------------------")
plt.plot(x, f)
plt.xlabel(' x ')
plt.ylabel(' F(x) ')
plt.title('RAMP FUNCTION')
```

```
plt.show()
```

OUTPUT:

```
------------------------------------------------------------------
                        RAMP FUNCTION
------------------------------------------------------------------
Enter input value X : -32 -27 -15 -7 0 9 18 27 36
------------------------------------------------------------------
INPUT (X) : -32  -27  -15  -7  0  9  18  27  36
------------------------------------------------------------------
OUTPUT : 0  0  0  0  0  1  1  1  1
------------------------------------------------------------------
```

PLOT:



Conclusion: Thus, we have successfully implemented Activation Functions of Neural Network. Understood the concept behind the activation functions. Plotted graph for each of the implemented activation functions.

Post Lab Descriptive Questions :

1. Explain the concept behind using Activation function.

ANS) Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron. We know, neural network has neurons that work in correspondence of weight, bias and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as back-propagation. Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points. So, a neural network is a complex mesh of artificial neurons that imitates how the brain works. They take in input parameters with its associated weights and biases then we compute the weighted sum of the 'activated' neurons. Our activation function gets to decide which neurons will push forward the values into the next layer. In the forward propagation steps, our artificial neurons receive inputs from different parameters or features. Ideally, each input has its own value or weight and biases which can display interdependency that leads to change in the final prediction value. This is a phenomenon referred to as the Interaction effect. A good example would be when trying a regression model on a dataset of diabetic patients. The goal is to predict if a person runs the risk of diabetes based on their body weight and height. Some bodyweights indicate a greater risk of diabetes if a person has a shorter height compared to a taller person who relatively has better health index. There are of course other parameters which we are not considering at the moment. We say there is an interaction effect between height and bodyweight. The activation function takes into account the interaction effects in different parameters and does a transformation after which it gets to decide which neuron passes forward the value into the next layer. We learned the numerous activation functions that are used in ML models. For researchers, these functions are used to draw a comparison of what works best given the problem statement. There is no hard and fast rule for selecting a particular activation function. However, it depends upon the model's architecture, the hyperparameters and the features that we are attempting to capture.

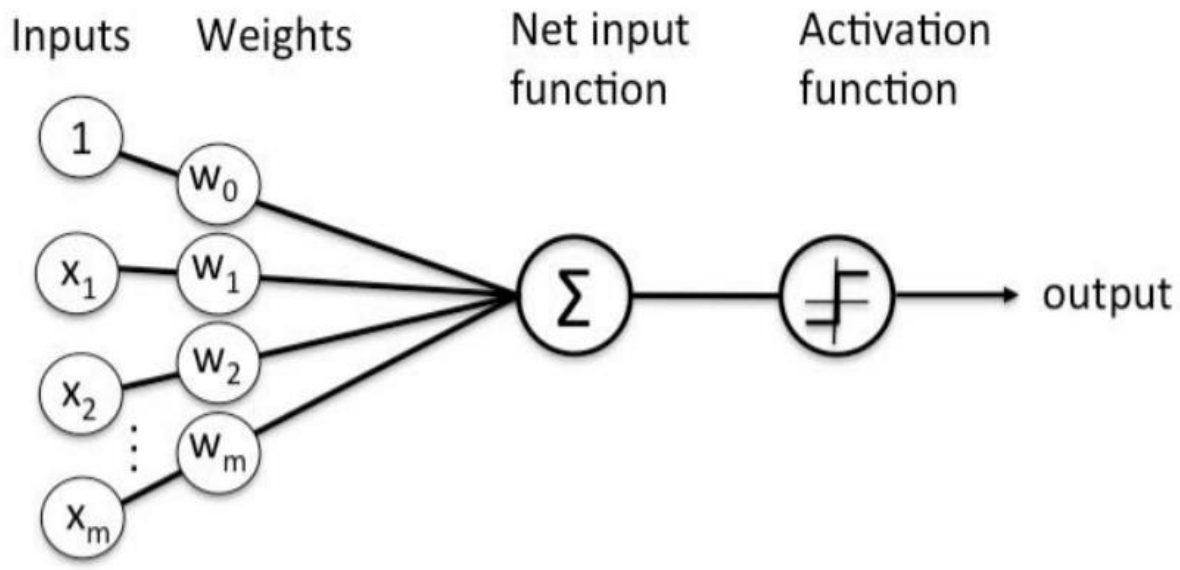
2. Explain different types of neural network.

ANS) Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another. Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Types of neural network:

A. Perceptron

Perceptron model, proposed by Minsky-Papert is one of the simplest and oldest models of Neuron. It is the smallest unit of neural network that does certain computations to detect features or business intelligence in the input data. It accepts weighted inputs, and apply the activation function to obtain the output as the final result. Perceptron is also known as TLU(threshold logic unit) Perceptron is a supervised learning algorithm that classifies the data into two categories, thus it is a binary classifier.

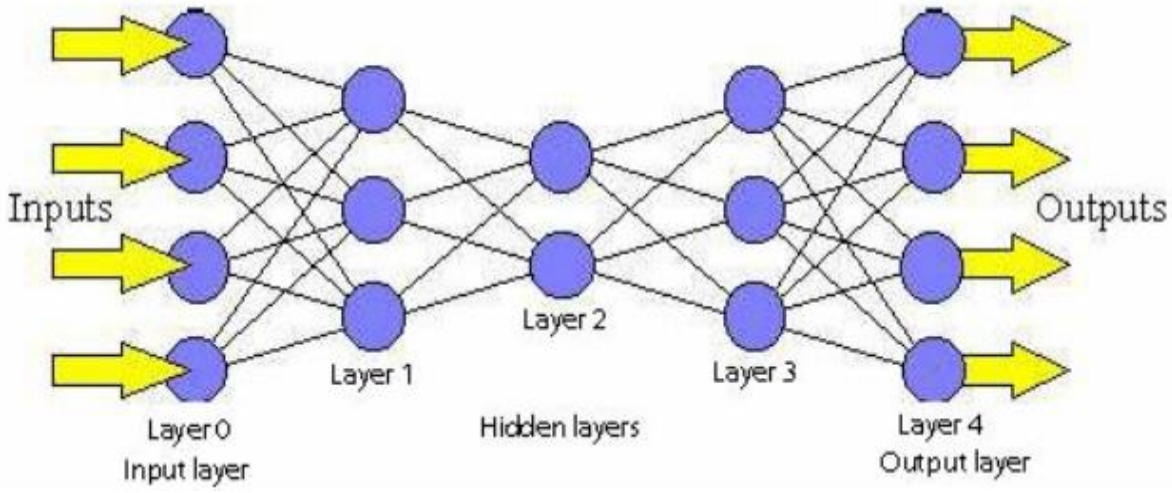


Perceptron has the following characteristics:

- Perceptron is an algorithm for Supervised Learning of single layer binary linear classifiers.
- Optimal weight coefficients are automatically learned.
- Weights are multiplied with the input features and decision is made if the neuron is fired or not.
- Activation function applies a step rule to check if the output of the weighting function is greater than zero.
- Linear decision boundary is drawn enabling the distinction between the two linearly separable classes +1 and -1.
- If the sum of the input signals exceeds a certain threshold, it outputs a signal; otherwise, there is no output.

B. Feed Forward Neural Networks

The simplest form of neural networks where input data travels in one direction only, passing through artificial neural nodes and exiting through output nodes. Where hidden layers may or may not be present, input and output layers are present there. Based on this, they can be further classified as a single-layered or multi-layered feed-forward neural network. Number of layers depends on the complexity of the function. It has unidirectional forward propagation but no backward propagation. Weights are static here. An activation function is fed by inputs which are multiplied by weights. To do so, classifying activation function or step activation function is used. For example: The neuron is activated if it is above threshold (usually 0) and the neuron produces 1 as an output. The neuron is not activated if it is below threshold (usually 0) which is considered as -1. They are fairly simple to maintain and are equipped with to deal with data which contains a lot of noise.

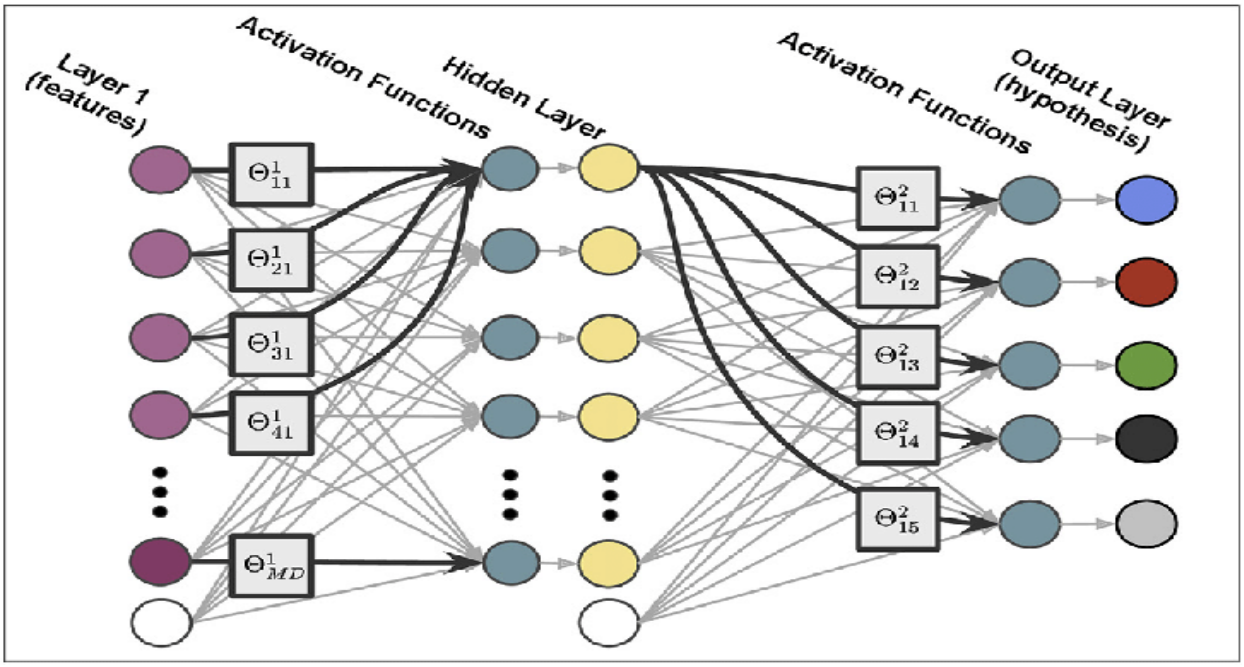


Feed-forward networks have the following characteristics:

- Perceptrons are arranged in layers, with the first layer taking in inputs and the last layer producing outputs. The middle layers have no connection with the external world, and hence are called hidden layers.
- Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is constantly "fed forward" from one layer to the next., and this explains why these networks are called feed-forward networks.
- There is no connection among perceptrons in the same layer.

C. Multilayer Perceptron

An entry point towards complex neural nets where input data travels through various layers of artificial neurons. Every single node is connected to all neurons in the next layer which makes it a fully connected neural network. Input and output layers are present having multiple hidden Layers i.e. at least three or more layers in total. It has a bi-directional propagation i.e. forward propagation and backward propagation. Inputs are multiplied with weights and fed to the activation function and in backpropagation, they are modified to reduce the loss. In simple words, weights are machine learnt values from Neural Networks. They self-adjust depending on the difference between predicted outputs vs training inputs. Nonlinear activation functions are used followed by softmax as an output layer activation function.

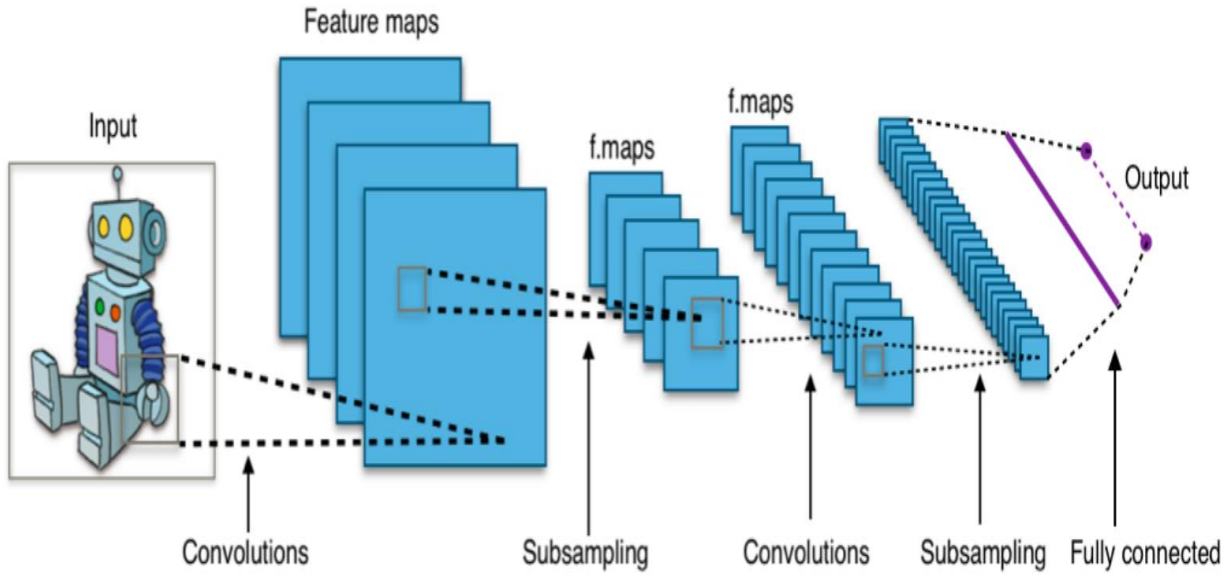


Multilayer perceptron has the following characteristics:

- The model of each neuron in the network incudes a nonlinear activation function. The nonlinearity is smooth and differentiation everywhere by using sigmoidal nonlinearity.
- The network contains one or more layers of hidden neurons which enable the network to learn complex tasks by extracting more meaningful features from the input patterns.
- The network exhibits a high degree of connectivity, determined by the synapses of the network.

D. Convolutional Neural Network

Convolution neural network contains a three-dimensional arrangement of neurons, instead of the standard two-dimensional array. The first layer is called a convolutional layer. Each neuron in the convolutional layer only processes the information from a small part of the visual field. Input features are taken in batch-wise like a filter. The network understands the images in parts and can compute these operations multiple times to complete the full image processing. Processing involves conversion of the image from RGB or HSI scale to grey-scale. Furthering the changes in the pixel value will help to detect the edges and images can be classified into different categories. Propagation is unidirectional where CNN contains one or more convolutional layers followed by pooling and bidirectional where the output of convolution layer goes to a fully connected neural network for classifying the images as shown in the above diagram. Filters are used to extract certain parts of the image. In MLP the inputs are multiplied with weights and fed to the activation function. Convolution uses RELU and MLP uses nonlinear activation function followed by softmax. Convolution neural networks show very effective results in image and video recognition, semantic parsing and paraphrase detection.

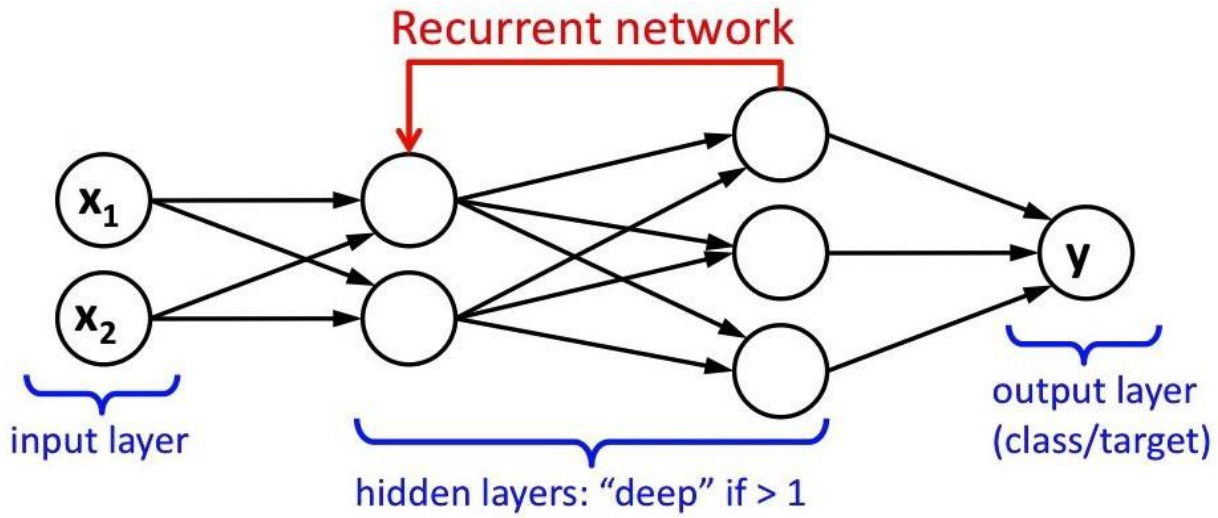


Convolutional neural networks has the following characteristics:

- Convolutional neural network is a class of deep learning methods which has become dominant in various computer vision tasks and is attracting interest across a variety of domains, including radiology.
- Convolutional neural network is composed of multiple building blocks, such as convolution layers, pooling layers, and fully connected layers, and is designed to automatically and adaptively learn spatial hierarchies of features through a backpropagation algorithm.
- Familiarity with the concepts and advantages, as well as limitations, of convolutional neural network is essential to leverage its potential to improve radiologist performance and, eventually, patient care.

E. Recurrent Neural Networks:

Designed to save the output of a layer, Recurrent Neural Network is fed back to the input to help in predicting the outcome of the layer. The first layer is typically a feed forward neural network followed by recurrent neural network layer where some information it had in the previous time-step is remembered by a memory function. Forward propagation is implemented in this case. It stores information required for it's future use. If the prediction is wrong, the learning rate is employed to make small changes. Hence, making it gradually increase towards making the right prediction during the backpropagation. Model sequential data where each sample can be assumed to be dependent on historical ones is one of the advantage. Used with convolution layers to extend the pixel effectiveness. Gradient vanishing and exploding problems. Training recurrent neural nets could be a difficult task. Difficult to process long sequential data using ReLU as an activation function.



Recurrent neural networks has the following characteristics:

- An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short Term Memory.
- Recurrent neural network are even used with convolutional layers to extend the effective pixel neighborhood.
- The error is then back-propagated to the network to update the weights and hence the network (RNN) is trained.

Date: 28 / 8 / 2021 **Signature of faculty in-charge**