EXPERIMENT NO. 2

AIM: Implement different activation functions in Artificial Neural Network.

SOFTWARE: Python

LABORATORY OUTCOMES:

- Students will be able implement different activation functions in neural network.
- Students will be able to calculate output of neural network for given activation functions.

THEORY:

The activation function is used to calculate the output response of a neuron. The sum of the weighted input signal is applied with an activation to obtain the response. For neuron in same layer, same activation functions are used. There may be linear and non-linear activation functions. The nonlinear activation functions are used in multilayer net. Few activation functions are explained below,

1. Binary Step Activation Function:

Binary step function is a threshold-based activation function which means after a certain threshold neuron is activated and below the said threshold neuron is deactivated. This activation function can be used in binary classifications as the name suggests, however it can not be used in a situation where you have multiple classes to deal with.

Binary step function returns value either 0 or 1. It returns '0' if the input is the less than zero. It returns '1' if the input is greater than zero.

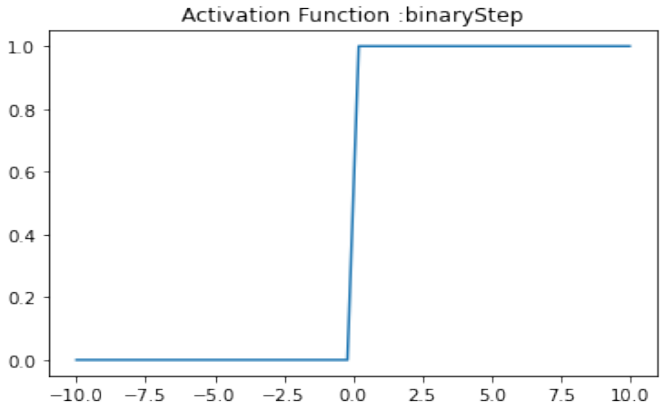


Fig. 2.1 Binary step activation function

2. Linear Activation Function:

The Linear Activation Function is given by $f(x) = x$. The graph of Linear Activation Function for values of 'x' is as shown in fig. 2.2. These functions are pretty simple. It returns what it gets as input. Here, output is directly proportional to the weighted sum of neurons. This function can deal with multiple classes, unlike Binary Step function. However, it has its own drawbacks. With this function changes made in back-propagation will be constant which is not good for learning. Another huge drawback of this function is that no matter how deep the neural network is (how many layers' neural

network consist of) last layer will always be a function of the first layer. This limits the neural network's ability to deal with complex problems.

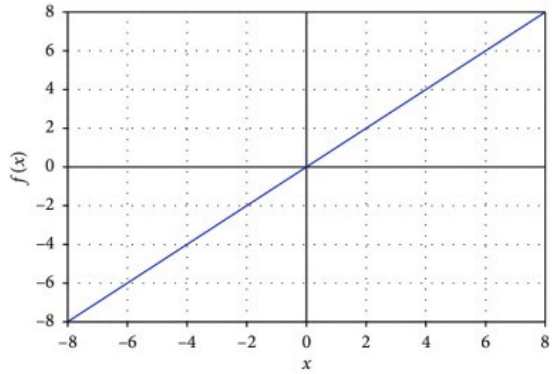


Fig. 2.2 Linear activation function

Non-Linear Activation Functions:

Deep learning practitioners today work with data of high dimensionality such as images, audios, videos, etc. With the drawbacks mentioned above, it is not practical to use Linear Activation Functions in complex applications that we use neural networks for. Therefore, it is Non-Linear Functions that are being widely used in present.

3. Sigmoid Activation Function:

Sigmoid function also known as logistic function, it is a function or logistic curve is a common "S" shape (sigmoid curve) as shown in fig. 2.3.

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

Sigmoid function returns the value between 0 and 1. It normalizes the output of each neuron. However, Sigmoid function makes almost no change in the prediction for very high or very low inputs which ultimately results in neural network refusing to learn further, this problem is known as the *vanishing gradient*.

The logistic function finds applications in a range of fields, including artificial neural networks, biology (especially ecology), biomathematics, chemistry, demography, economics, geoscience, mathematical psychology, probability, sociology, political science, linguistics, and statistics.

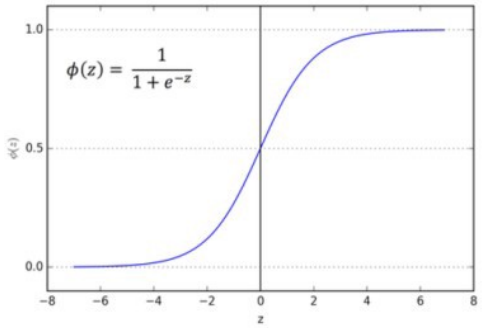


Fig. 2.3 Sigmoid Activation Function

4. Bipolar Sigmoid Activation Function:

A bipolar sigmoid function is of the form **f(x)=- 1+2/(1+exp(-x))**. The range of values of sigmoid functions can be varied depending on the application. However, the range of (-1,+1) is most commonly adopted. The Bipolar sigmoid function is as shown in fig. 2.4

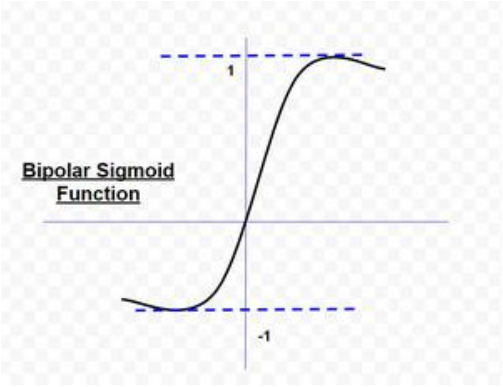


Fig. 2.4 Bipolar Sigmoid Activation Function

4. Hyperbolic Tangent Function (tanh function):

Hyperbolic tangent function (also known as tanh) is almost like the sigmoid function but slightly better than that since it's output ranges between -1 and 1 allowing negative outputs. However, 'tanh' also comes with the *vanishing gradient* problem just like sigmoid function. Hyperbolic tangent function is as shown in fig. 2.5.

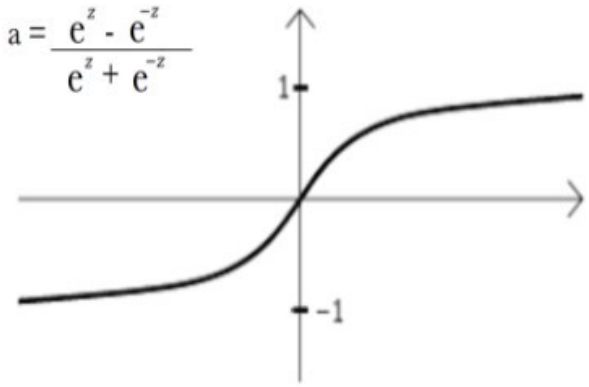


Fig. 2.5 Hyperbolic Tangent Function

5. ReLU (Rectified Linear Unit) Function:

In this function, outputs for the positive inputs can range from 0 to infinity but when the input is zero or a negative value, the function outputs zero and it hinders with the back-propagation. The ReLU activation function is as shown in fig. 2.6. This problem is known as the dying ReLU problem.

Mathematically it is represented as:

$$f(x) = \max(0, x) = \begin{cases} x_i \text{ if } x_i > 0 \\ 0 \text{ if } x_i < 0 \end{cases}$$

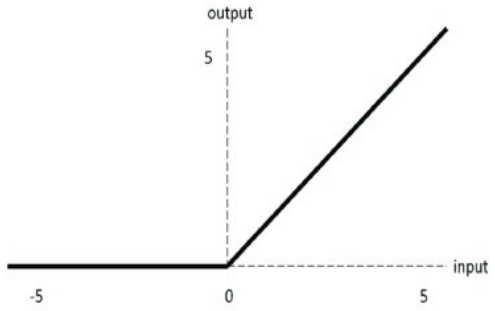


Fig. 2.6 ReLU Activation Function

One major benefit is the reduced likelihood of the gradient to vanish. This arises when a>0a>0. In this regime the gradient has a constant value. In contrast, the gradient of sigmoid becomes increasingly small as the absolute value of 'x' increases. The constant gradient of ReLUs results in faster learning.

6. Leaky ReLU Activation Function:

Leaky Rectified Linear Unit**, or** Leaky ReLU**,** is a type of activation function based on a ReLU, but it has a small slope for negative values instead of a flat slope. Leaky ReLU function is as shown in fig. 2.7. The slope coefficient is determined before training, i.e. it is not learnt during training. This type of activation function is popular in tasks where we may suffer from sparse gradients, for example training generative adversarial networks.

Mathematical representation:

$$f(x)=\max(0.01*x,x)=\begin{cases} x, \text{ if } x>0 \\ 0.01*x, \text{ if } x<0 \end{cases}$$

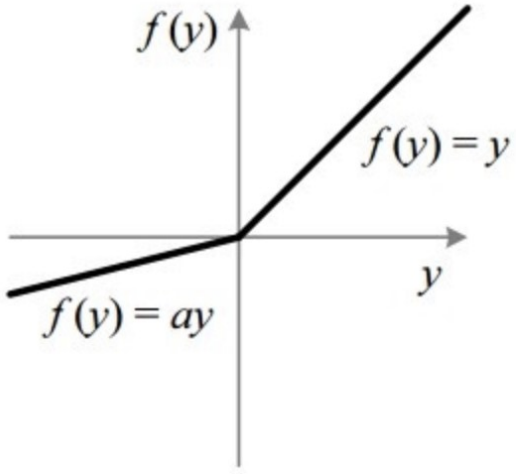


Fig. 2.7 Leaky ReLU Activation Function

7. Softmax Activation Function:

The softmax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution. Softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels. The output of a Softmax is a vector with probabilities of each possible outcome. The probabilities in a softmax vector, sums to one for all possible outcomes or classes. The softmax activation function is as shown in fig. 2.8.

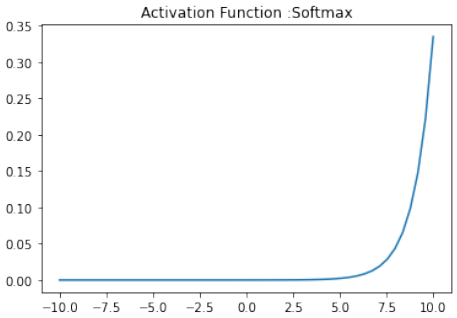


Fig. 2.8 Softmax Activation Function

Mathematically, Softmax is defined as,

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

where,

y	is an input vector to a softmax function, S. It consist of n elements for n classes (possible outcomes)
y_i	the i-th element of the input vector. It can take any value between -inf and +inf
$exp(y_i)$	standard exponential function applied on y_i. The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg • $\exp(55) = 7.69e{+}23$ (A very large value) • $\exp(-55) = 1.30e\text{-}24$ (A very small value close to 0) **Note**: $\exp(*)$ is just $\mathbf{e}^*$ where $\mathbf{e} = 2.718$, the Euler's number.
$\sum_{j=1}^{n} \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sums to 1 for i-th class and each of them and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
# Binary Step Activation Function
def binaryStep(x):
    ''' It returns '0' is the input is less then zero otherwise it returns one '''
    return np.heaviside(x,1)
x = np.linspace(-10, 10)
plt.plot(x, binaryStep(x))
plt.axis('tight')
plt.title('Activation Function :BinaryStep')
plt.show()
# Linear Activation Function
def linear(x):
    ''' y = f(x) It returns the input as it is'''
    return x
x = np.linspace(-10, 10)
plt.plot(x, linear(x))
plt.axis('tight')
plt.title('Activation Function :Linear')
plt.show()
# Sigmoid Activation Function
def sigmoid(x):
    ''' It returns 1/(1+exp(-x)). where the values lies between zero and one '''
    return 1/(1+np.exp(-x))
x = np.linspace(-10, 10)
plt.plot(x, sigmoid(x))
plt.axis('tight')
plt.title('Activation Function :Sigmoid')
plt.show()
#Bipolar Sigmoid Activation Function
def bipolarsigmoid(x):
    ''' It returns -1+2/(1+exp(-x)). where the values lies between zero and one '''
    return -1+2/(1+np.exp(-x))
x = np.linspace(-10, 10)
```

```
plt.plot(x, bipolarsigmoid(x))
plt.axis('tight')
plt.title('Activation Function :BipolarSigmoid')
plt.show()
```

 # Tanh Activation Function

```
def tanh(x):
    ''' It returns the value (1-exp(-2x))/(1+exp(-2x)) and the value returned will be lies in between -1 to
1.'''
    return np.tanh(x)
x = np.linspace(-10, 10)
plt.plot(x, tanh(x))
plt.axis('tight')
plt.title('Activation Function :Tanh')
plt.show()
```

 # ReLU Activation Function

```
def RELU(x):
    ''' It returns zero if the input is less than zero otherwise it returns the given input. '''
    x1=[]
    for i in x:
        if i<0:
            x1.append(0)
        else:
            x1.append(i)
    return x1
x = np.linspace(-10, 10)
plt.plot(x, RELU(x))
plt.axis('tight')
plt.title('Activation Function :RELU')
plt.show()
```

#Leaky ReLU Activation Function

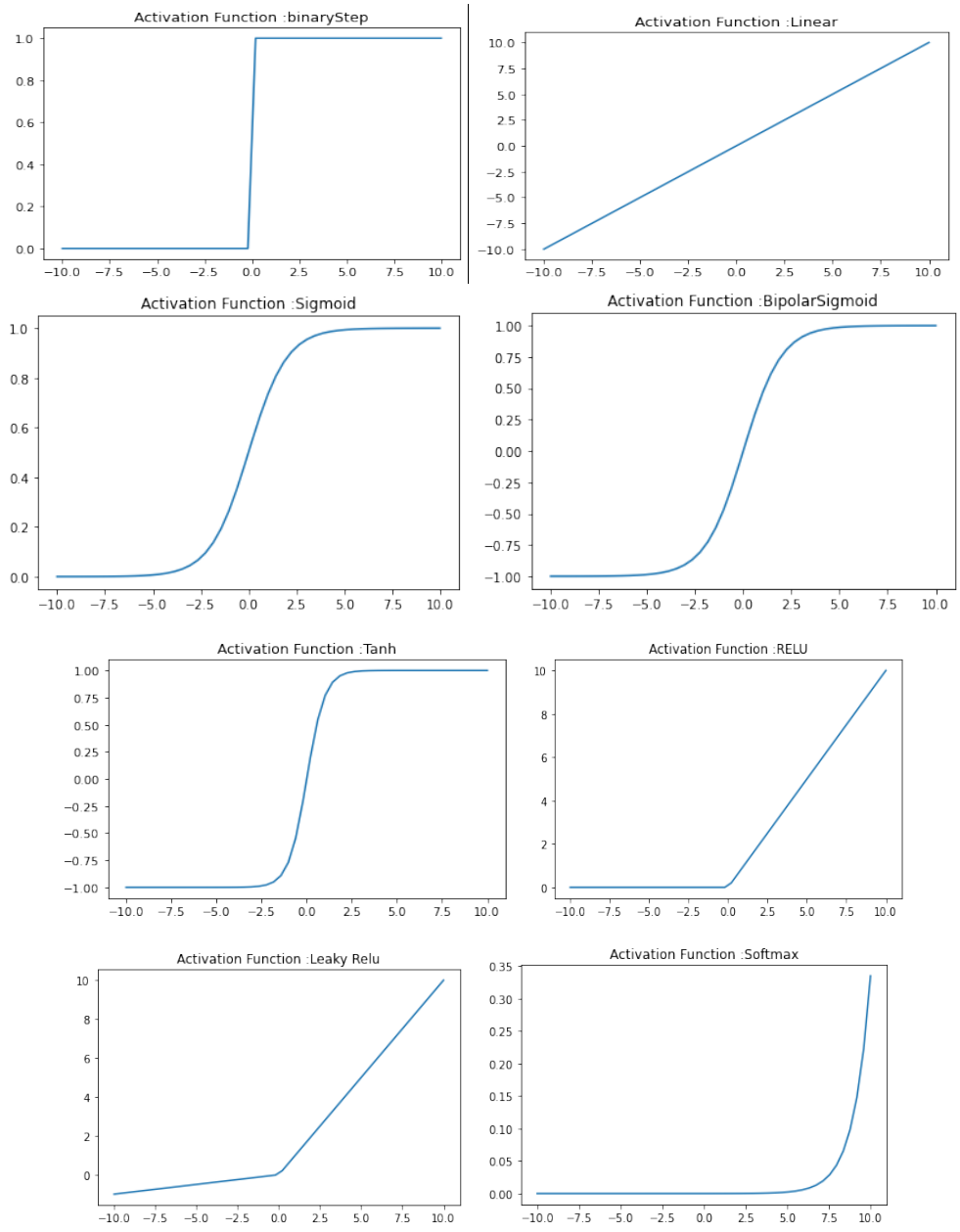
```
def leaky_relu(x):
    alpha = 0.1
    return np.maximum(alpha*x, x)
x = np.linspace(-10, 10)
plt.plot(x, leaky_relu(x))
plt.axis('tight')
```

```
plt.title('Activation Function :Leaky Relu')
plt.show()
```

 # Softmax Activation Function

```
def softmax(x):
    ''' Compute softmax values for each sets of scores in x. '''
    return np.exp(x) / np.sum(np.exp(x), axis=0)
x = np.linspace(-10, 10)
plt.plot(x, softmax(x))
plt.axis('tight')
plt.title('Activation Function :Softmax')
plt.show()
```

OUTPUTS:



CONCLUSION:

The activation functions play a major role in determining the output of the functions. There are linear and non-linear activation functions. Nonlinear activation function is used in multilayer neural network. Thus for the given neural network with the given weights and bias we will be able to calculate output for the given activation function.

TEXT/REFERENCE BOOKS:

- "Neural Network a Comprehensive Foundation" By Simon Haykin
- "Introduction to Soft Computing" By Dr. S. N. Shivnandam, Mrs. S. N. Deepa
- "Neural Network: A classroom Approach" By Satish Kumar
- "Neural Network, Fuzzy Logic and Genetic Algorithms" By Rajshekharan S, Vijayalakshmi Pai
- "Neural Network Design" by Hagan Demuth, Beale
- "Neural Network for Pattern Recognition", Christopher M. Bishop

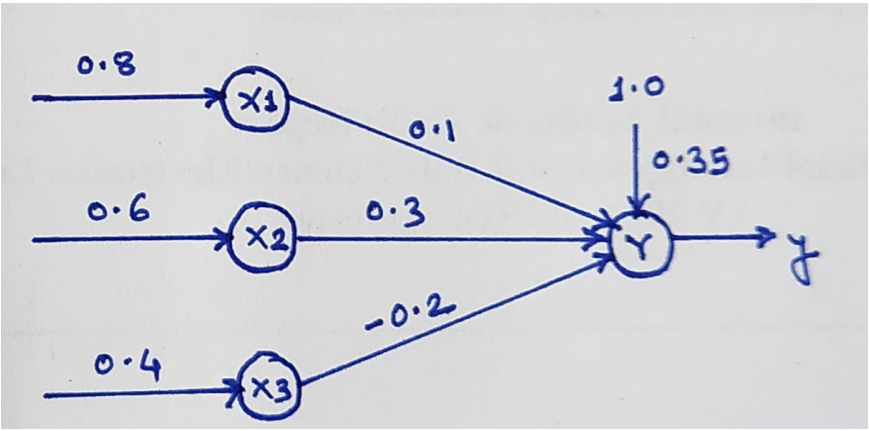
WEB ADDRESS URL:

- https://en.wikipedia.org/wiki/Activation_function
- https://analyticsindiamag.com/most-common-activation-functions-in-neural-networks

EXERCISE:

1. Obtain the output of the neuron Y for the network shown using activation function.

a) Binary sigmoidal b) Bipolar sigmoidal c) Hyperbolic tangent



----------x----------x----------