Batch: A1 Roll No.: 16010120015

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.

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.

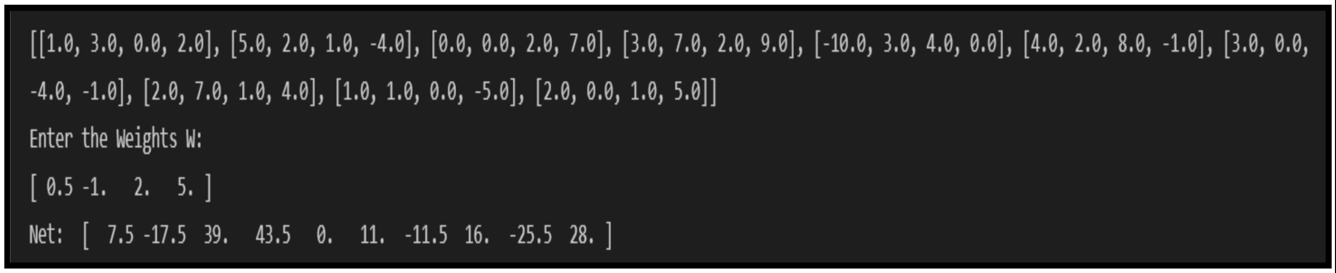
Implementation Details:

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 :

```
import numpy as np
import matplotlib.pyplot as plt

def linear(x):
    return x


def sigmoid(x):
    sigout = 1.0 / (1.0 + np.exp(-x))
    return sigout


def bipolar_sigmoidal(x):
    bisigout = (1.0 - np.exp(-x)) / (1.0 + np.exp(-x))
    return bisigout


def step(x):
    thresh = float(input("Threshold value: "))
    stepout=[]
    for value in x:
        if value >= thresh:
            stepout.append(1)
        else:
            stepout.append(0)
    return stepout
```

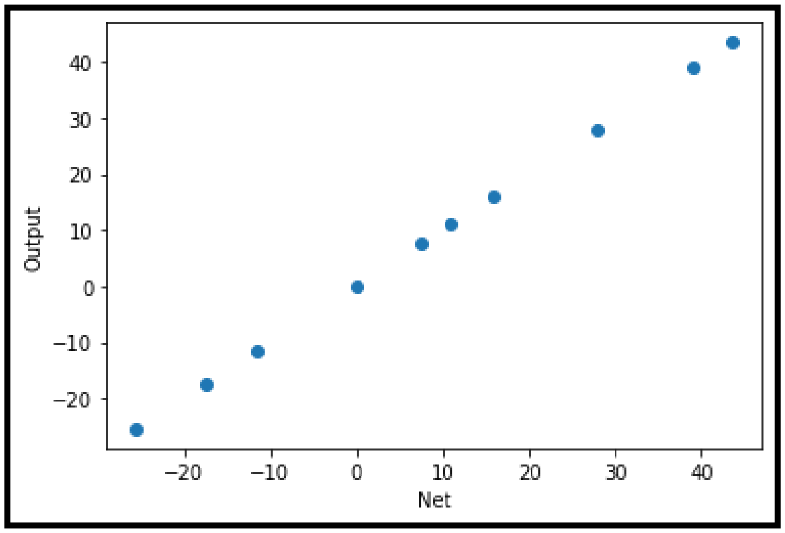
```python
inputArr = []
i = 0
while i<10:


    print("Enter the Input X: ")
    x = list(map(float, input().split()))
    x = inputArr.append(x)
    i += 1

print(inputArr)


print("Enter the Weights W: ")
w = list(map(float, input().split()))
w = np.array(w)

print(w)

netArr = np.dot(inputArr,w)


print("Net: ", netArr)

while True:

    print("Choose an activation function: ")
    print("1. Identity")
    print("2. Sigmoid")
    print("3. bipolarSigmoid")
    print("4. Threshold")
    print("5. Exit")
    choice = int(input())
    if choice == 1:
        out = linear(netArr)
        plt.scatter(netArr,out)
        plt.xlabel("Net")
        plt.ylabel("Output")
        plt.show()
    elif choice == 2:
        out = sigmoid(netArr)
        print(out)
        plt.scatter(netArr,out)
        plt.xlabel("Net")
        plt.ylabel("Output")
        plt.show()
    elif choice == 3:
        out = bipolar_sigmoidal(netArr)
        plt.scatter(netArr,out)
        plt.xlabel("Net")
        plt.ylabel("Output")
        plt.show()
    elif choice == 4:
        out = step(netArr)
        plt.scatter(netArr,out)
        plt.xlabel("Net")
        plt.ylabel("Output")
        plt.show()
    elif choice == 5:
        break
    else:
        out = 0
        print("Invalid choice")

    print("Output: ", out)
```

Input vector



```
[[1.0, 3.0, 0.0, 2.0], [5.0, 2.0, 1.0, -4.0], [0.0, 0.0, 2.0, 7.0], [3.0, 7.0, 2.0, 9.0], [-10.0, 3.0, 4.0, 0.0], [4.0, 2.0, 8.0, -1.0], [3.0, 0.0,
-4.0, -1.0], [2.0, 7.0, 1.0, 4.0], [1.0, 1.0, 0.0, -5.0], [2.0, 0.0, 1.0, 5.0]]
Enter the Weights W:
[ 0.5 -1.   2.   5. ]
Net:  [  7.5 -17.5  39.   43.5   0.   11.  -11.5  16.  -25.5  28. ]
```

1) **Identity function:** The Identity function is also known as *linear activation Function* where the activation is proportional to the input.

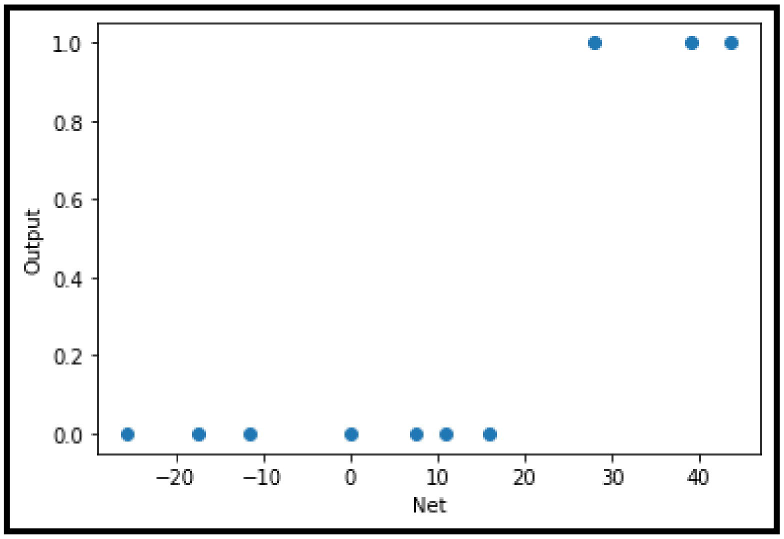


2) **Binary step function:** Binary step function depends on a threshold value that decides whether a neuron should be activated or not.

The input fed to the activation function is compared to a certain threshold; if the input is greater than it, then the neuron is activated, else it is deactivated, meaning that its output is not passed on to the next hidden layer.

Binary step

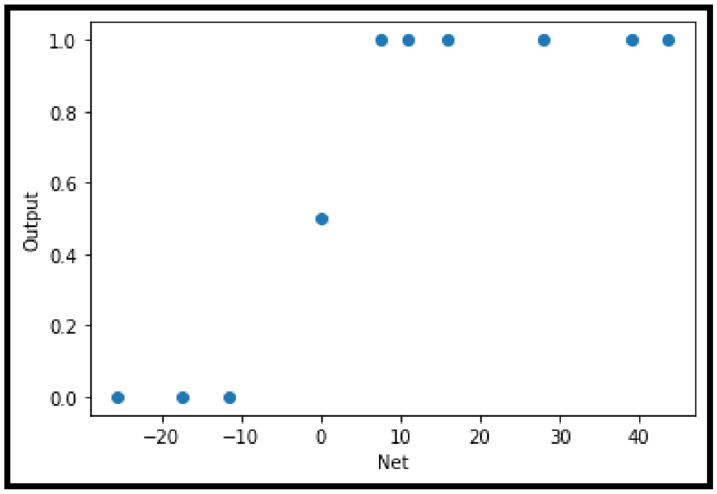
$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$



3) Sigmoid function: This function takes any real value as input and outputs values in the range of 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0, as shown below.

Sigmoid / Logistic

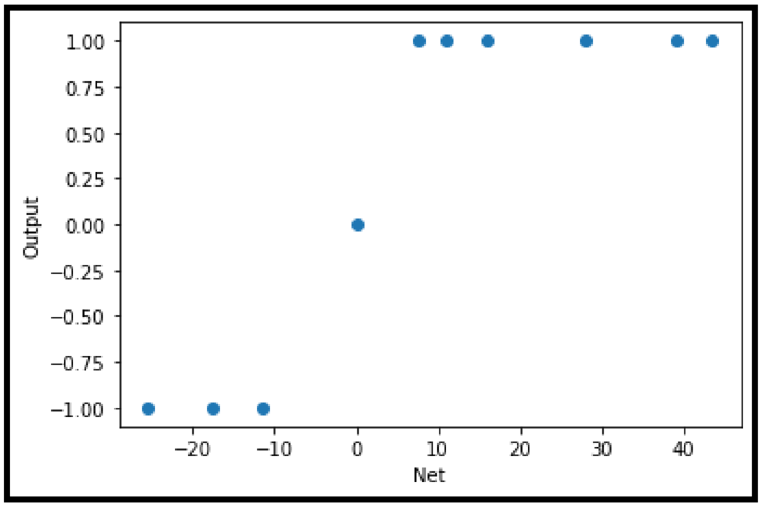
$$f(x) = \frac{1}{1 + e^{-x}}$$



4) Bipolar sigmoid function: Tanh function is very similar to the sigmoid/logistic activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

Tanh

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



Conclusion:

In this, we tried explaining all the non linear activation functions with the mathematical expressions Thus, we have successfully implemented 4 Activation Functions of Neural Network.

Post Lab Descriptive Questions :

1. Explain the concept behind using Activation function.

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.

Explanation :- 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. hence we can conclude that, doesn't matter how many hidden layer we attach in neural net, all layers will behave same way because *the composition of two linear function is a linear function itself*. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. Hence we need activation function

2. Explain the different properties of activation functions.

Properties of activation functions

1. Non Linearity
2. Continuously differentiable
3. Range
4. Monotonic
5. Approximates identity near the origin

1**)Non Linearity**- The purpose of the activation function is to introduce non-linearity into

the network in turn allows you to model a response variable (aka target variable, class

label, or score) that varies non-linearly with its explanatory variables

2) **Continuously differentiable** - This property is necessary for enabling gradient-based

optimization methods.

3) **Approximates identity near the origin -** When activation functions have this property, the neural network will learn efficiently when its weights are initialized with small random values.

4**) Monotonic** - When the activation function is monotonic, the error surface associated with a single-layer model is guaranteed to be convex.

5**) Range** - When the range of the activation function is finite, gradient-based training methods tend to be more stable, because pattern presentations significantly affect only limited weights.

Date: ____10/10/2022___

Signature of faculty in-charge