Ex. No 5. PERCEPTRON DESIGN TO EMPLOY THE SIGMOID ACTIVATION FUNCTION

Aim:

To develop a python code that creates a simple feed-forward neural networks or perception with the Sigmoid activation function. The neuron has to be trained such that it can predict the correct output value when provided with a new set of input data.

	Input			Output
Training data 1	0	0	1	0
Training data 2	1	1	1	1
Training data 3	1	0	1	1
Training data 4	0	1	1	0
New Situation	1	0	0	**?**

Equipments Required:

1. Hardware – PCs
2. Anaconda – Python 3.7 Installation / Moodle-Code Runner / Google Colab.

Related Theoritical Concept:

Sigmoid Activation Function is commonly used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.

The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function.

The limitations of sigmoid function are :

The derivative of the function is f'(x) = sigmoid(x)*(1-sigmoid(x)).

The output of the logistic function is not symmetric around zero. So the output of all the neurons will be of the same sign. This makes the training of the neural network more difficult and unstable.

Algorithm:

1.Import packages.

2.Define neural network class.

3.Model single neuron with 3 inputs and 1 output and assign random weights to a 3 x 1 matrix with values between -1 and 1.

4.Define sigmoid function and calculate the error.

5.Multiply the error by the input and again by the gradient of the sigmoid curve.

6.Initialize a single neuron neural network.

7.Train neural network using training data.

8.Test neural network with new data.

Program:

/*

Program to implement the sigmoid activation function in a feed forward ANN.

Developed by: S.SUDHAKARAN

Register Number: 212219040155

***/**

```
import numpy as np
class NeuralNetwork():
    def___init_(self):
        np.random.seed(1)
        self.synaptic_weights=2*np.random.random((3,1)) -1

    def  sigmoid(self,x):
        return 1/(1+np.exp(-x))

    def sigmoid_derivative(self,x):
        return x*(1-x)

    def train(self,training_inputs,training_outputs,training_iterations):

        for iteration in range(training_iterations):
```

```
            output=self.think(training_inputs)
            error=training_outputs-output
            adjustments=np.dot(training_inputs.T,error*self.sigmoid_derivative(output))
            self.synaptic_weights += adjustments
    def think(self,inputs):
        inputs=inputs.astype(float)
        output=self.sigmoid(np.dot(inputs,self.synaptic_weights))
        return output

if__name__=="_main_":
    neuron=NeuralNetwork()

    print("Beginning Randomly Generated weights: ")
    print(neuron.synaptic_weights)

    training_inputs=np.array([[0,0,1],[1,1,1],[1,0,1],[0,1,1]])

    training_outputs=np.array([[0,1,1,0]]).T

    neuron.train(training_inputs,training_outputs, 15000)

    print("Ending Weights After Training")
    print(neuron.synaptic_weights)

    user_input_one=str(input("User Input One : "))
    user_input_two=str(input("User Input Two : "))
    user_input_three=str(input("User Input Three : "))

    print("Considering New Situation: ",user_input_one, user_input_two,
user_input_three)
    print("New Output data: ")
    print(neuron.think(np.array([user_input_one, user_input_two, user_input_three])))
    print("Wow, we did it!")
```

Output:

```
Beginning Randomly Generated weights:
[[-0.16595599]
 [ 0.44064899]
 [-0.99977125]]
Ending Weights After Training
[[10.08740896]
 [-0.20695366]
 [-4.83757835]]
User Input One : 6
User Input Two : 7
User Input Three : 8
Considering New Situation:  6 7 8
New Output data:
[1.]
Wow, we did it!
```

Result:

Thus created a perception to employ the Sigmoid activation function. This neuron was successfully trained to predict the correct output value, when provided with a new set of input data.