University Institute of Engineering

Department of Electronics & Communication Engineering

Experiment No. :- 1.2

Student Name: Priyanshu Mathur	**UID:** 20BEC1073
Branch: Electronics and Communication	**Section/Group:** A
Semester: 7th	**Date of Performance:** 17/08/23
Subject Name: AI & ML	**Subject Code:** 20ECA-445

1. Aim of the practical: Write a program to study the effect of different parameters on ANN.

2. Tool Used: Google Colab

3. Theory:

The network architecture with 10 inputs and 1 output. There are 10 inputs, X1 to X10 and 10 weights, W1 to W10. Each input Xi has its weight Wi.

A Perceptron is an artificial neuron. It is the simplest possible Neural Network.
Neural Networks are the building blocks of machine learning.

$$SOP = X_1W_1+X_2W_2+X_3W_3+X_4W_4+X_5W_5+X_6W_6+X_7W_7+X_8W_8+X_9W_9+X_{10}W_{10}$$

If we take the feedback known as backward pass we have to calculate error by subtracting the target value and predicted value for all 10 inputs and sending them back to the input side. There the change will be introduced to the weights of the inputs.

We have to calculate the rate of change of error with respect to weights.

$$\frac{dError}{dW_1} = \frac{dError}{dPredicted}\frac{dPredicted}{dSOP}\frac{dSOP}{dW_1}$$

To update the weights after each iteration for every inputs:-

$$W_{new} = W_{old} - learning\_rate * grad$$

Here,
Grad is rate of change of error with respect to weights.
Learning rate is the factor by which we are multiplying the grad to get new accurate weight.

4. Steps for experiment/practical:

Step 1:- Defining and implementing the necessary function.

Step 2:- Defining input, output, weight or bias, and learning rate.

Step 3 :- Implementing the forward pass.

Step 4 :- If needed implementing backward pass.

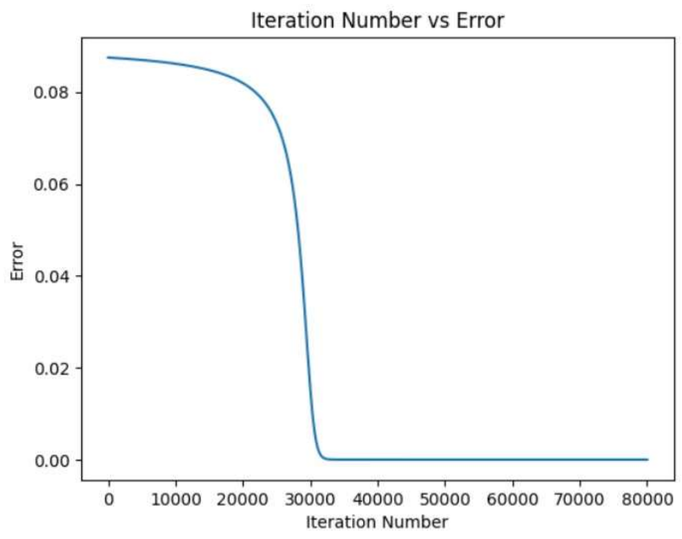
Step 5 :- Run and execute the program.

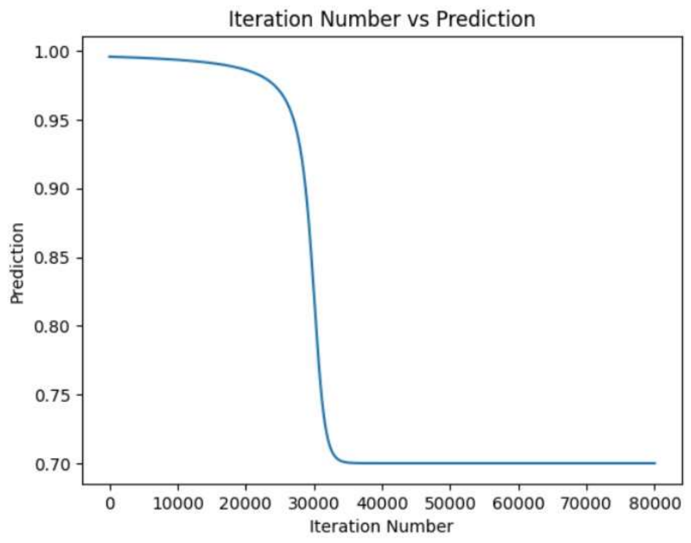
5. Program Code and Simulation Output:

Code:-

```
import numpy

import matplotlib.pyplot

def sigmoid(sop):

    return 1.0/(1+numpy.exp(-1*sop))

def error(predicted, target):

    return numpy.power(predicted-target, 2)

def error_predicted_deriv(predicted, target):

    return 2*(predicted-target)

def sigmoid_sop_deriv(sop):
    return sigmoid(sop)*(1.0-sigmoid(sop))
def sop_w_deriv(x):
    return x
def update_w(w, grad, learning_rate):
    return w - learning_rate*grad


values = [0.1, 0.4, 1.1, 1.3, 1.8, 2.0, 0.01, 0.9, 0.8, 1.6]
row_matrix = numpy.array(values)
target = 0.7
learning_rate = 0.0009
num_weights = 10
weights = numpy.array([numpy.random.rand() for _ in range(num_weights)])
print("Initial Weights:")
print(weights)
predicted_output = []
network_error = []
old_err = 0
for k in range(80000):
    y= numpy.dot(values,weights)
    predicted = sigmoid(y)
    err = error(predicted, target)
    predicted_output.append(predicted)
    network_error.append(err)
    # Backward Pass
    g1 = error_predicted_deriv(predicted, target)
    g2 = sigmoid_sop_deriv(y)
    for i in range(num_weights):
        g3_weights = numpy.array([sop_w_deriv(x) for x in values])
        gradw = g3_weights[i] * g2 * g1
        weights[i] = update_w(weights[i], gradw, learning_rate)
    print(predicted)
matplotlib.pyplot.figure()
matplotlib.pyplot.plot(network_error)
matplotlib.pyplot.title("Iteration Number vs Error")
matplotlib.pyplot.xlabel("Iteration Number")
matplotlib.pyplot.ylabel("Error")

matplotlib.pyplot.figure()
matplotlib.pyplot.plot(predicted_output)
matplotlib.pyplot.title("Iteration Number vs Prediction")
matplotlib.pyplot.xlabel("Iteration Number")
matplotlib.pyplot.ylabel("Prediction")
```

Simulation Output: -





Result and Discussion: -

In this experiment we have designed a perceptron with ten input and a single output. We have done a large number of iterations to reach our target value by updating the weights and learning rate through backward pass that is first processed and then sent to the activation function. In this experiment, I have observed that in comparison to single input perceptron, the multiple input perceptron reaches the target value earlier with less iteration. The error and prediction also corrects itself in very less iteration in comparison to there calculation for single input.

Learning outcomes (What I have learnt):

- Learnt about the structure of multiple input perceptron.
- Learnt how the forward pass and backward pass works for multiple input structure.
- Learnt about the error calculation and relative error that has to be calculated for multiple inputs at the same time.
- Learnt about the learning rate and its effect on the output.
- Observed the results of multiple inputs to SOP and activation function and the results of the single input to SOP and activation function and compared them.