Ex: 2 b

BUILD A SIMPLE NEURAL NETWORKS

Aim:

To build a simple neural network without Keras/TensorFlow.

Algorithm:

1. Initialize Parameters such as layer sizes, weights and biases.

2. Use sigmoid activation function and its derivative for backpropagation.

3. Forward pass: Compute activations for the hidden and output layers using weights, biases and sigmoid.

4. Calculate Error: Find the difference between predicted output and target values.

5. Backpropagate: Calculate gradients for weights and biases using the error and sigmoid derivative.

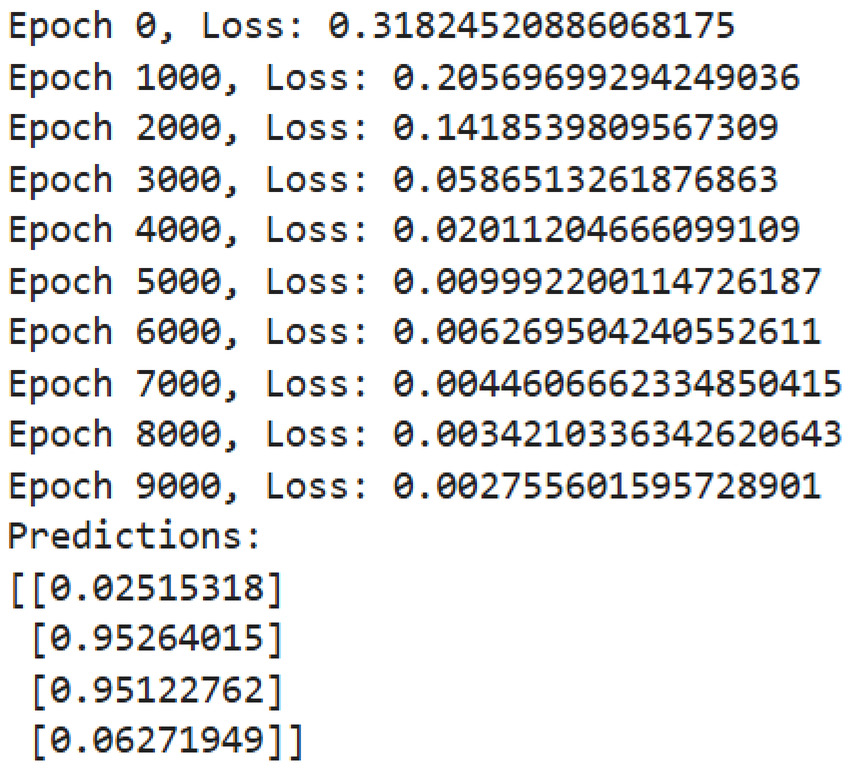
6. Update Parameters: Adjust weights and biases using the gradients and learning rate, and repeat for multiple epochs.

Program:

```
import numpy as np

# Define the size of each layer

input_size = 2

hidden_size = 3

output_size = 1

# Initialize weights with random values

np.random.seed(42)

W1 = np.random.randn(input_size, hidden_size)  # weights between input and hidden layer

W2 = np.random.randn(hidden_size, output_size)  # weights between hidden and output layer

# Initialize biases with zeros

b1 = np.zeros((1, hidden_size))  # bias for hidden layer

b2 = np.zeros((1, output_size))  # bias for output layer

def sigmoid(x):

    return 1 / (1 + np.exp(-x))
```

```
def sigmoid_derivative(x):
    return x * (1 - x)

def forward_pass(X):
    z1 = np.dot(X, W1) + b1
    a1 = sigmoid(z1)
    z2 = np.dot(a1, W2) + b2
    a2 = sigmoid(z2)
    return a1, a2

def backward_pass(X, y, a1, a2):
    global W1, W2, b1, b2
    # Calculate the error
    error = y - a2
    d2 = error * sigmoid_derivative(a2)
    # Calculate the gradient for W2 and b2
    dW2 = np.dot(a1.T, d2)
    db2 = np.sum(d2, axis=0, keepdims=True)
    # Calculate the gradient for W1 and b1
    d1 = np.dot(d2, W2.T) * sigmoid_derivative(a1)
    dW1 = np.dot(X.T, d1)
    db1 = np.sum(d1, axis=0, keepdims=True)
    # Update weights and biases
    W1 += learning_rate * dW1
    b1 += learning_rate * db1
    W2 += learning_rate * dW2
    b2 += learning_rate * db2

def train(X, y, epochs, learning_rate):
    global W1, W2, b1, b2
    for epoch in range(epochs):
        a1, a2 = forward_pass(X)
        backward_pass(X, y, a1, a2)
        if epoch % 1000 == 0:
            loss = np.mean(np.square(y - a2))
            print(f'Epoch {epoch}, Loss: {loss}')

# Input data: XOR problem
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[0], [1], [1], [0]])
# Hyperparameters
learning_rate = 0.1
epochs = 10000
# Train the network
train(X, y, epochs, learning_rate)
# Test the network
_, output = forward_pass(X)
print("Predictions:")
print(output)
```

Output:



```
Epoch 0, Loss: 0.31824520886068175
Epoch 1000, Loss: 0.20569699294249036
Epoch 2000, Loss: 0.1418539809567309
Epoch 3000, Loss: 0.0586513261876863
Epoch 4000, Loss: 0.02011204666099109
Epoch 5000, Loss: 0.009992200114726187
Epoch 6000, Loss: 0.006269504240552611
Epoch 7000, Loss: 0.0044606662334850415
Epoch 8000, Loss: 0.0034210336342620643
Epoch 9000, Loss: 0.002755601595728901
Predictions:
[[0.02515318]
 [0.95264015]
 [0.95122762]
 [0.06271949]]
```

Result:

Thus the program to build a simple neural network without Keras/TensorFlow has been executed successfully.