Ex No: 2b **Date: 02-08-2024**

BUILD A SIMPLE NEURAL NETWORK WITHOUT KERAS

AIM:

To build a simple neural network from scratch using NumPy to solve the XOR problem.

PROCEDURE:

1. Create XOR Dataset: Define the XOR input-output pairs.
2. Initialize Parameters: Set up the network structure and randomly initialize weights and biases.
3. Implement Functions: Define sigmoid activation and its derivative.
4. Forward Propagation: Compute network outputs using the activation function.
5. Backpropagation: Update weights and biases by propagating error backward using gradient descent.
6. Train Network: Iterate forward and backward propagation for a set number of epochs.
7. Evaluate Performance: Predict XOR outputs and compare with actual results.

PROGRAM:

```
import numpy as np

def sigmoid(x):

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

def sigmoid_derivative(x):

    return x * (1 - x)

X = np.array([[0, 0],

              [0, 1],

              [1, 0],

              [1, 1]])

y = np.array([[0],

              [1],

              [1],

              [0]])
```

```
input_layer_neurons = X.shape[1]   # Number of input features
hidden_layer_neurons = 2           # Number of neurons in the hidden layer
output_neurons = 1                 # Number of neurons in the output layer
weights_input_hidden = np.random.uniform(size=(input_layer_neurons, hidden_layer_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_layer_neurons, output_neurons))
bias_hidden = np.random.uniform(size=(1, hidden_layer_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))
learning_rate = 0.1
epochs = 10000
for epoch in range(epochs):
    hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
    hidden_layer_activation = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_activation, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(weights_hidden_output.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_activation)
    weights_hidden_output += hidden_layer_activation.T.dot(d_predicted_output) * learning_rate
    bias_output += np.sum(d_predicted_output, axis=0, keepdims=True) * learning_rate
    weights_input_hidden += X.T.dot(d_hidden_layer) * learning_rate
    bias_hidden += np.sum(d_hidden_layer, axis=0, keepdims=True) * learning_rate
    if epoch % 1000 == 0:
        loss = np.mean(np.abs(error))
        print(f'Epoch {epoch}, Loss: {loss:.4f}')
```

print("Predicted Output after training:")

print(predicted_output)

OUTPUT:

```
Epoch 0, Loss: 0.4993
Epoch 1000, Loss: 0.4995
Epoch 2000, Loss: 0.4972
Epoch 3000, Loss: 0.4795
Epoch 4000, Loss: 0.4146
Epoch 5000, Loss: 0.3491
Epoch 6000, Loss: 0.1874
Epoch 7000, Loss: 0.1162
Epoch 8000, Loss: 0.0878
Epoch 9000, Loss: 0.0724
Predicted Output after training:
[[0.06444773]
 [0.93970551]
 [0.93969475]
 [0.06577409]]
```

RESULT:

Thus, a simple neural network using NumPy was successfully implemented to solve the XOR problem.