Feedforward Backpropagation Neural Network

Aim:

1. Implement the Feed Forward Backpropagation algorithm to train a neural network.
2. Use a given dataset to train the neural network for a specific task.
3. Evaluate the performance of the trained network on test data

Theory: Feedforward neural networks, also known as feedforward neural networks (FNNs) or multilayer perceptrons (MLPs), are a fundamental type of artificial neural network used in machine learning and deep learning. They are designed to model complex relationships between inputs and outputs by stacking multiple layers of interconnected neurons (also called nodes or units).

Here are the key characteristics of feedforward neural networks:

1. Feedforward Structure: FNNs have a strict feedforward structure, meaning information flows in one direction, from the input layer through one or more hidden layers to the output layer. There are no feedback loops or recurrent connections in this architecture.

2. Layers: An FNN typically consists of three main types of layers:
 - Input Layer: This layer contains neurons that represent the features or input data. Each neuron corresponds to a specific input feature.
 - Hidden Layers: These intermediate layers, which can be one or more, perform complex transformations on the input data. Each neuron in a hidden layer is connected to all neurons in the previous layer and feeds its output to the next layer.
 - Output Layer: The final layer produces the network's predictions or outputs. The number of neurons in the output layer depends on the problem; for regression tasks, it may be one neuron, while for classification tasks, it can be one neuron per class.

3. Activation Functions: Non-linear activation functions (e.g., ReLU, sigmoid, or tanh) are applied to the output of each neuron in the hidden layers. These functions introduce non-linearity into the network, enabling it to capture complex patterns in the data.

4. Weights and Biases: Every connection between neurons has an associated weight that determines the strength of the connection. Additionally, each neuron has a bias term that helps shift the activation function. These weights and biases are learned during training to optimize the network's performance.

5. Training: FNNs are trained using supervised learning methods, such as gradient descent and backpropagation. The network is presented with labelled training data, and it adjusts its weights and biases to minimize the difference between its predictions and the actual target values.

6. Loss Function: A loss function (also called a cost or objective function) quantifies the error between the network's predictions and the actual target values. The goal during training is to minimize this loss function.

7. Applications: Feedforward neural networks are used in a wide range of applications,

including image and speech recognition, natural language processing, financial modeling, and many other machine learning tasks.

8. Deep Learning: When FNNs have multiple hidden layers, they are referred to as deep neural networks (DNNs). Deep learning leverages these deep architectures to model intricate relationships in data, making them suitable for handling complex and high-dimensional problems.

In summary, feedforward neural networks are a foundational component of deep learning, allowing machines to learn and make predictions from data by forming increasingly abstract and hierarchical representations through multiple layers of interconnected neurons.

For the present case we implement the XOR- operation using Feedforward Backpropagation Neural Network. The XOR-operation for a 2-input variable is as follows

Inputs		Target
X1	X0	T
0	0	0
0	1	1
1	0	1
1	1	0

The following is the Python code to implement Feedforward Backpropagation Neural Network

Python Code

```
import numpy as np
# Define the sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
# Define the neural network class
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize weights with random values
        self.weights_input_hidden = np.random.uniform(-1, 1, (input_size, hidden_size))
        self.weights_hidden_output = np.random.uniform(-1, 1, (hidden_size, output_size))

    def forward(self, inputs):
        # Forward propagation
        self.hidden_input = np.dot(inputs, self.weights_input_hidden)
        self.hidden_output = sigmoid(self.hidden_input)
        self.output_input = np.dot(self.hidden_output, self.weights_hidden_output)
        self.predicted_output = sigmoid(self.output_input)
        return self.predicted_output

    def backward(self, inputs, target, learning_rate):
        # Backpropagation
        error = target - self.predicted_output
        delta_output = error * sigmoid_derivative(self.predicted_output)

        error_hidden = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = error_hidden * sigmoid_derivative(self.hidden_output)

        # Update weights
```

```
        self.weights_hidden_output += np.outer(self.hidden_output, delta_output) *
learning_rate
        self.weights_input_hidden += np.outer(inputs, delta_hidden) * learning_rate

    def train(self, training_data, targets, epochs, learning_rate):
        for epoch in range(epochs):
            for i in range(len(training_data)):
                inputs = training_data[i]
                target = targets[i]
                self.forward(inputs)
                self.backward(inputs, target, learning_rate)

    def predict(self, inputs):
        return self.forward(inputs)

# Define XOR dataset
training_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
targets = np.array([[0], [1], [1], [0]])

# Create and train the neural network
input_size = 2
hidden_size = 4
output_size = 1
learning_rate = 0.1
epochs = 10000

nn = NeuralNetwork(input_size, hidden_size, output_size)
nn.train(training_data, targets, epochs, learning_rate)

# Test the trained network
for i in range(len(training_data)):
    inputs = training_data[i]
    prediction = nn.predict(inputs)
    print(f"Input: {inputs}, Predicted Output: {prediction}")
```

Output

```
Input: [0 0], Predicted Output: [0.16943356]
Input: [0 1], Predicted Output: [0.8562007]
Input: [1 0], Predicted Output: [0.85655176]
Input: [1 1], Predicted Output: [0.12905042]
```

We conclude that the Predicted Output is very close to the Target

For Video demonstration of the practical click on the link or scan the QR-code

https://youtu.be/ILS919-KlDo