EXPERIMENT 6

AIM : WRITE A PROGRAM TO IMPLEMENT A NEURAL NETWORK IN PYTHON

THEORY : Neural networks are a foundational theory in artificial intelligence that mimic the structure and function of the human brain to process data and learn patterns. They consist of layers of interconnected nodes (neurons) that transform input data through weighted connections and activation functions. As data passes through these layers, the network learns to make predictions or classifications by adjusting weights via training algorithms like backpropagation. Neural networks power many modern AI applications, including image recognition, natural language processing, and autonomous systems.

CODE :

```
import numpy as np


# Activation function (Sigmoid)
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivative of Sigmoid
def sigmoid_derivative(x):
    return x * (1 - x)


# Neural Network class
class NeuralNetwork:
    def _init_(self, input_nodes, hidden_nodes, output_nodes):
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes


        # Initialize weights and biases
        self.weights_input_hidden = np.random.rand(self.input_nodes,
self.hidden_nodes)
        self.weights_hidden_output = np.random.rand(self.hidden_nodes,
self.output_nodes)
```

```python
        self.bias_hidden = np.random.rand(self.hidden_nodes)
        self.bias_output = np.random.rand(self.output_nodes)

    def feedforward(self, X):
        # Forward propagation
        self.hidden_layer_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_layer_output = sigmoid(self.hidden_layer_input)

        self.final_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) + self.bias_output
        self.final_output = sigmoid(self.final_input)
        return self.final_output

    def train(self, X, y, learning_rate, epochs):
        for _ in range(epochs):
            # Forward pass
            self.feedforward(X)

            # Backpropagation
            output_error = y - self.final_output
            output_delta = output_error * sigmoid_derivative(self.final_output)

            hidden_error = np.dot(output_delta, self.weights_hidden_output.T)
            hidden_delta = hidden_error * sigmoid_derivative(self.hidden_layer_output)

            # Update weights and biases
            self.weights_hidden_output += np.dot(self.hidden_layer_output.T, output_delta) * learning_rate
            self.bias_output += np.sum(output_delta, axis=0) * learning_rate
```

```
        self.weights_input_hidden += np.dot(X.T, hidden_delta) * learning_rate
        self.bias_hidden += np.sum(hidden_delta, axis=0) * learning_rate

# Example usage
if _name_ == "_main_":
    # Sample dataset (X: inputs, y: outputs)
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    y = np.array([[0], [1], [1], [0]])  # XOR problem

    nn = NeuralNetwork(input_nodes=2, hidden_nodes=4, output_nodes=1)
    nn.train(X, y, learning_rate=0.5, epochs=10000)

    # Test the network
    print("Predictions:")
    print(nn.feedforward(X))
```

```
Predictions:
[[0.01502373]
 [0.98154121]
 [0.98974501]
 [0.01589742]]
```