**16. Write the python program to implement Feed forward neural Network**

**Program:**

import numpy as np

# Sigmoid activation function and its derivative

def sigmoid(x):

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

def sigmoid\_deriv(x):

return x \* (1 - x)

# Training data (XOR logic gate)

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

[0, 1],

[1, 0],

[1, 1]])

# Output labels for XOR

y = np.array([[0],

[1],

[1],

[0]])

# Set random seed for reproducibility

np.random.seed(42)

# Initialize weights and biases

input\_size = 2

hidden\_size = 4

output\_size = 1

weights\_input\_hidden = np.random.uniform(size=(input\_size, hidden\_size))

bias\_hidden = np.random.uniform(size=(1, hidden\_size))

weights\_hidden\_output = np.random.uniform(size=(hidden\_size, output\_size))

bias\_output = np.random.uniform(size=(1, output\_size))

# Training the network

epochs = 10000

learning\_rate = 0.1

for epoch in range(epochs):

# Forward pass

hidden\_input = np.dot(X, weights\_input\_hidden) + bias\_hidden

hidden\_output = sigmoid(hidden\_input)

final\_input = np.dot(hidden\_output, weights\_hidden\_output) + bias\_output

final\_output = sigmoid(final\_input)

# Calculate loss (MSE)

loss = y - final\_output

# Backpropagation

d\_output = loss \* sigmoid\_deriv(final\_output)

error\_hidden\_layer = d\_output.dot(weights\_hidden\_output.T)

d\_hidden = error\_hidden\_layer \* sigmoid\_deriv(hidden\_output)

# Update weights and biases

weights\_hidden\_output += hidden\_output.T.dot(d\_output) \* learning\_rate

bias\_output += np.sum(d\_output, axis=0, keepdims=True) \* learning\_rate

weights\_input\_hidden += X.T.dot(d\_hidden) \* learning\_rate

bias\_hidden += np.sum(d\_hidden, axis=0, keepdims=True) \* learning\_rate

# Print loss every 1000 epochs

if epoch % 1000 == 0:

print(f"Epoch {epoch}, Loss: {np.mean(np.square(loss)):.4f}")

# Final output

print("\nPredictions after training:")

print(final\_output)

**output:**

