```
Name: Aarya Admane Rollno: 22630 B2

import     numpy as    np

#    Sigmoid    activation function   and   its   derivative
def   sigmoid(x):
 return    1    /    (1    +    np.exp(-x))
def   sigmoid_derivative(x):
 return    x    *    (1    -    x)

 #   Input data  for   XOR   function
 X   =    np.array([[0,   0],  [0,   1],  [1,   0],  [1,   1]])
 #   Output     data  (labels    for   XOR   function)
 y   =    np.array([[0],  [1], [1], [0]])

 #   Seed  random     numbers    for   consistent results
 np.random.seed(42)

#    Initialize weights    randomly   with  mean  0
input_layer_neurons   =    2
hidden_layer_neurons  =    2
output_neurons   =    1

#    Random     weight     initialization
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  initialization
 bias_hidden    =    np.random.uniform(size=(1,
     hidden_layer_neurons))
 bias_output    =    np.random.uniform(size=(1, output_neurons))

 #   Training   parameters
 learning_rate  =    0.5
 epochs    =    10000

#    Training   process
for   epoch in    range(epochs):
 #   Forward    Propagation
 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)

 #   Compute    error
 error     =    y    -    final_output
```

```python
# Backpropagation
d_output = error * sigmoid_derivative(final_output)
error_hidden_layer = d_output.dot(weights_hidden_output.T)
d_hidden_layer = error_hidden_layer *
    sigmoid_derivative(hidden_output)

# Update weights and biases
weights_hidden_output += hidden_output.T.dot(d_output) *
    learning_rate
weights_input_hidden += X.T.dot(d_hidden_layer) *
    learning_rate
bias_output += np.sum(d_output, axis=0, keepdims=True) *
    learning_rate
bias_hidden += np.sum(d_hidden_layer, axis=0,
    keepdims=True) * learning_rate

# Print error every 1000 epochs
if epoch % 1000 == 0:
    print(f'Epoch {epoch}, Error: {np.mean(np.abs(error))}')

# Testing the trained network
print("\nFinal Output after training:")
print(final_output)
```

```
Final Output after training:
[[0.68808925]
 [0.69744107]
 [0.69633921]
 [0.7033354 ]]
```