```python
# Title: Implement Artificial Neural Network training process in Python
# by using Forward Propagation, Back Propagation for solved example from Sivanadam

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)

# Input dataset
inputs = np.array([0, 1])
expected_output = np.array([1])

# Initialize weights with random values
epochs = 5
lr = 0.25
inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2, 2, 1

hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))

# Training algorithm
for _ in range(epochs):
  # Forward Propagation
  # Forward Propagation
  hidden_layer_activation = np.dot(inputs,hidden_weights)
  # Reshape hidden_layer_activation to have the same number of dimensions as hidden_bias
  hidden_layer_activation = hidden_layer_activation.reshape(1, -1)  # Reshape to (1, 2)
  hidden_layer_activation += hidden_bias
  hidden_layer_output = sigmoid(hidden_layer_activation)

  output_layer_activation = np.dot(hidden_layer_output,output_weights)
  output_layer_activation += output_bias
  predicted_output = sigmoid(output_layer_activation)

  # Backpropagation
  error = expected_output - predicted_output
  d_predicted_output = error * sigmoid_derivative(predicted_output)

  error_hidden_layer = d_predicted_output.dot(output_weights.T)
  d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

  # Updating Weights and Biases
  output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
  output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
  hidden_layer_activation = hidden_layer_activation.reshape(1, -1)
  hidden_weights += inputs.T.dot(d_hidden_layer) * lr
  hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr

  output_weights += hidden_layer_output.T.dot(d_predicted_output) * lr
  output_bias += np.sum(d_predicted_output,axis=0,keepdims=True) * lr
  hidden_layer_activation = hidden_layer_activation.reshape(1, -1)
  hidden_weights += inputs.T.dot(d_hidden_layer) * lr # Use inputs.T for correct matrix multiplicatio
  hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) * lr

print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
```

```
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)

print("\nOutput from neural network after 10,000 epochs: ",end='')
print(*predicted_output)
```

```
Final hidden weights: [0.23692198 0.26817839] [0.12161607 0.9638183 ]
Final hidden bias: [0.96059838 0.74253138]
Final output weights: [0.55301649] [0.3712982]
Final output bias: [0.76496477]

Output from neural network after 2 epochs: [0.81397937]
Final hidden weights: [0.23692198 0.26817839] [0.12233516 0.96415042]
Final hidden bias: [0.96131747 0.74286349]
Final output weights: [0.5581543] [0.37712009]
Final output bias: [0.77184349]

Output from neural network after 2 epochs: [0.81641914]
Final hidden weights: [0.23692198 0.26817839] [0.12304394 0.96447992]
Final hidden bias: [0.96202625 0.74319299]
Final output weights: [0.56317714] [0.38281022]
Final output bias: [0.77856585]

Output from neural network after 2 epochs: [0.81877954]
Final hidden weights: [0.23692198 0.26817839] [0.12374262 0.96480677]
Final hidden bias: [0.96272493 0.74351984]
Final output weights: [0.56808954] [0.38837381]
Final output bias: [0.78513805]

Output from neural network after 2 epochs: [0.82106444]
Final hidden weights: [0.23692198 0.26817839] [0.12443146 0.96513094]
Final hidden bias: [0.96341376 0.74384402]
Final output weights: [0.57289579] [0.39381581]
Final output bias: [0.79156596]

Output from neural network after 2 epochs: [0.82327746]
```