```
import numpy as np



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

# Forward propagation
def forward(X, weights):
  z = np.dot(X, weights)  # Linear combination
  return sigmoid(z)       # Apply activation function
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
weights = np.random.randn(2, 1)
output = forward(X, weights)
print("Predicted Output:\n", output)
```

```
Predicted Output:
 [[0.5       ]
 [0.19585705]
 [0.6684618 ]
 [0.32934351]]
```

Compute the loss

```
def mean_squared_error(y_true, y_pred):
  return np.mean((y_true - y_pred) ** 2)
# True labels (logical AND function)
y_true = np.array([[0], [0], [0], [1]])

loss = mean_squared_error(y_true, output)
print("Loss:", loss)
```

```
Loss: 0.29624532259304637
```

Backpropogtion

```
def sigmoid_derivative(x):
  return x * (1 - x)
# Backpropagation function
def backpropagate(X, y_true, y_pred, weights, learning_rate=0.01):
# Output layer error
  error = y_true - y_pred
# Gradient for output layer (using chain rule)
  d_weights = np.dot(X.T, error * sigmoid_derivative(y_pred))
# Update the weights using the gradients
  weights += d_weights * learning_rate
  return weights
# Perform one step of backpropagation
weights = backpropagate(X, y_true, output, weights) ##


print(weights)
```

```
[[ 0.70123608]
 [-1.41121912]]
```

train the network

```
def train(X, y_true, weights, epochs=10000, learning_rate=0.01):
    for epoch in range(epochs):
        y_pred = forward(X, weights)

        # Backpropagation and weight update
        weights = backpropagate(X, y_true, y_pred, weights, learning_rate)

        # Print loss every 1000 epochs
        if epoch % 100 == 0:
            loss = mean_squared_error(y_true, y_pred)
            print(f'Epoch {epoch}, Loss: {loss}')

    return weights
```

```
# Train the network
weights = train(X, y_true, weights)
```

```
Epoch 4200, Loss: 0.25000000092886626
Epoch 4300, Loss: 0.2500000008196896
Epoch 4400, Loss: 0.2500000007233453
Epoch 4500, Loss: 0.25000000063832506
Epoch 4600, Loss: 0.2500000005632978
Epoch 4700, Loss: 0.25000000049708915
Epoch 4800, Loss: 0.25000000043866244
Epoch 4900, Loss: 0.2500000003871031
Epoch 5000, Loss: 0.25000000034160397
Epoch 5100, Loss: 0.2500000003014527
Epoch 5200, Loss: 0.25000000026602065
Epoch 5300, Loss: 0.2500000002347532
Epoch 5400, Loss: 0.25000000020716096
Epoch 5500, Loss: 0.2500000001828117
Epoch 5600, Loss: 0.25000000016132445
Epoch 5700, Loss: 0.2500000001423628
Epoch 5800, Loss: 0.2500000001256298
Epoch 5900, Loss: 0.25000000011086354
Epoch 6000, Loss: 0.25000000009783296
Epoch 6100, Loss: 0.25000000008633394
Epoch 6200, Loss: 0.2500000000761864
Epoch 6300, Loss: 0.25000000006723166
Epoch 6400, Loss: 0.2500000000593294
Epoch 6500, Loss: 0.25000000005235595
Epoch 6600, Loss: 0.25000000004620215
Epoch 6700, Loss: 0.2500000000407717
Epoch 6800, Loss: 0.2500000000359795
Epoch 6900, Loss: 0.2500000000317505
Epoch 7000, Loss: 0.2500000000280187
Epoch 7100, Loss: 0.25000000002472544
Epoch 7200, Loss: 0.25000000002181927
Epoch 7300, Loss: 0.25000000001925465
Epoch 7400, Loss: 0.2500000000169915
Epoch 7500, Loss: 0.25000000001499434
Epoch 7600, Loss: 0.25000000001323197
Epoch 7700, Loss: 0.2500000000116767
Epoch 7800, Loss: 0.25000000001030426
Epoch 7900, Loss: 0.25000000000909306
Epoch 8000, Loss: 0.25000000000802436
Epoch 8100, Loss: 0.25000000000708117
Epoch 8200, Loss: 0.25000000000624883
Epoch 8300, Loss: 0.25000000000551437
Epoch 8400, Loss: 0.2500000000048662
Epoch 8500, Loss: 0.2500000000042943
Epoch 8600, Loss: 0.2500000000037895
Epoch 8700, Loss: 0.2500000000033441
Epoch 8800, Loss: 0.2500000000029511
Epoch 8900, Loss: 0.25000000000260414
Epoch 9000, Loss: 0.25000000000229816
Epoch 9100, Loss: 0.25000000000202804
Epoch 9200, Loss: 0.2500000000017896
Epoch 9300, Loss: 0.2500000000015793
Epoch 9400, Loss: 0.25000000000139366
Epoch 9500, Loss: 0.25000000000122985
Epoch 9600, Loss: 0.2500000000010853
Epoch 9700, Loss: 0.25000000000095773
Epoch 9800, Loss: 0.2500000000008451
Epoch 9900, Loss: 0.25000000000074585
```

```
# Test the trained network
final_output = forward(X, weights)
print("Final Predicted Output:\n", final_output)
```

```
Final Predicted Output:
 [[0.5       ]
 [0.49999885]
 [0.50000115]
 [0.5       ]]
```

training wiht parameters

Generate | print hello world using rot13 | Close

```
import numpy as np
```

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

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

# Mean Squared Error (MSE) Loss function
def mean_squared_error(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)

# Forward propagation
def forward(X, weights):
    z = np.dot(X, weights)
    return sigmoid(z)

# Backpropagation function
def backpropagate(X, y_true, y_pred, weights, learning_rate=0.01):
    # Output layer error
    error = y_true - y_pred
    # Gradient for output layer (using chain rule)
    d_weights = np.dot(X.T, error * sigmoid_derivative(y_pred))
    # Update the weights using the gradients
    weights += d_weights * learning_rate
    return weights

# Training function
def train(X, y_true, weights, epochs=10000, learning_rate=0.01):
    for epoch in range(epochs):
        # Forward propagation
        y_pred = forward(X, weights)
        # Backpropagation and weight update
        weights = backpropagate(X, y_true, y_pred, weights, learning_rate)
        # Print loss every 1000 epochs
        if epoch % 1000 == 0:
            loss = mean_squared_error(y_true, y_pred)
            print(f'Epoch {epoch}, Loss: {loss}')
    return weights

# Main function
if __name__ == "__main__":
    # Input data (X) and weights
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input data (4 samples, 2 features)
    y_true = np.array([[0], [0], [0], [1]])  # True labels (logical AND function)
    weights = np.random.randn(2, 1)  # Random weights (2 input nodes to 1 output node)

    print("Initial Weights:\n", weights)

    # Train the network
    trained_weights = train(X, y_true, weights)

    # Test the trained network
    final_output = forward(X, trained_weights)
    print("Final Predicted Output:\n", final_output)

    # Experiment with hyperparameters
    print("\nExperimenting with learning rate:")
    trained_weights_lr = train(X, y_true, weights, learning_rate=0.1)
    final_output_lr = forward(X, trained_weights_lr)
    print("Final Predicted Output with higher learning rate:\n", final_output_lr)
```

```
Initial Weights:
 [[-0.69252755]
 [-1.4299286 ]]
Epoch 0, Loss: 0.2990156231183009
Epoch 1000, Loss: 0.2747801830417679
Epoch 2000, Loss: 0.252444429672653
Epoch 3000, Loss: 0.2501275760620719
Epoch 4000, Loss: 0.25001975469518517
Epoch 5000, Loss: 0.25000526382559923
Epoch 6000, Loss: 0.2500014988437737
Epoch 7000, Loss: 0.25000042908433523
Epoch 8000, Loss: 0.25000012288421747
Epoch 9000, Loss: 0.2500000351931991
Final Predicted Output:
 [[0.5       ]
```

```
 [0.49985805]
 [0.50014201]
 [0.50000005]]

Experimenting with learning rate:
Epoch 0, Loss: 0.25000001007908784
Epoch 1000, Loss: 0.2500000000000361
Epoch 2000, Loss: 0.25
Epoch 3000, Loss: 0.25
Epoch 4000, Loss: 0.25
Epoch 5000, Loss: 0.25
Epoch 6000, Loss: 0.25
Epoch 7000, Loss: 0.25
Epoch 8000, Loss: 0.25
Epoch 9000, Loss: 0.25
Final Predicted Output with higher learning rate:
 [[0.5]
 [0.5]
 [0.5]
 [0.5]]
```