ANN Using Back Propogation

```
import numpy as np

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

def sigmoid_derivative(x):
    return x * (1 - x)

# Backpropagation training algorithm
def train_backpropagation(X, y, hidden_neurons, epochs, learning_rate):
    input_neurons = X.shape[1]
    output_neurons = y.shape[1]

    # Initialize weights and biases
    weights_input_hidden = np.random.rand(input_neurons, hidden_neurons)
    weights_hidden_output = np.random.rand(hidden_neurons, output_neurons)
    bias_hidden = np.random.rand(hidden_neurons)
    bias_output = np.random.rand(output_neurons)

    for epoch in range(epochs):
        # Forward pass
        hidden_layer_input = np.dot(X, weights_input_hidden) + bias_hidden
        hidden_layer_output = sigmoid(hidden_layer_input)
```

```
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) +
bias_output
    predicted_output = sigmoid(output_layer_input)

    # Backward pass
    error = y - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)

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

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

    if epoch % 100 == 0:
        print(f"Epoch {epoch} error: {np.mean(np.abs(error))}")

  return weights_input_hidden, weights_hidden_output, bias_hidden, bias_output

# Example usage: training XNOR gate
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([[1], [0], [0], [1]])  # XNOR gate output

hidden_neurons = 2
```

```
epochs = 10000

learning_rate = 0.1


weights_input_hidden, weights_hidden_output, bias_hidden, bias_output =
train_backpropagation(
    X, y, hidden_neurons, epochs, learning_rate)


# Testing

def predict_backpropagation(input_data, weights_input_hidden, weights_hidden_output,
bias_hidden, bias_output):
    hidden_layer_input = np.dot(input_data, weights_input_hidden) + bias_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) + bias_output
    predicted_output = sigmoid(output_layer_input)
    return predicted_output


# Test the network with new inputs

test_input = np.array([1, 0])

prediction = predict_backpropagation(test_input, weights_input_hidden,
weights_hidden_output, bias_hidden, bias_output)

print(f"Prediction for {test_input}: {prediction}")
```