```python
import numpy as np

#define the sigmoid activation function,
# which is commonly used in neural networks. It takes an input x and returns the sigmoid function's output.

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# define the derivative of the sigmoid function, which is used during backpropagation for calculating gradients.
def sigmoid_derivative(x):
    return x * (1 - x)


class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.weights_input_hidden = np.random.rand(input_size, hidden_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.weights_hidden_output = np.random.rand(hidden_size, output_size)
        self.bias_output = np.zeros((1, output_size))

    def forward(self, inputs):
        self.inputs = inputs
        self.hidden_layer_input = np.dot(inputs, self.weights_input_hidden) + self.bias_hidden
        self.hidden_layer_output = sigmoid(self.hidden_layer_input)
        self.output_layer_input = np.dot(self.hidden_layer_output, self.weights_hidden_output) + self.bias_output
        self.output = sigmoid(self.output_layer_input)
        return self.output

    def backward(self, target, learning_rate):
        loss = target - self.output
        delta_output = loss * sigmoid_derivative(self.output)
        loss_hidden = delta_output.dot(self.weights_hidden_output.T)
        delta_hidden = loss_hidden * sigmoid_derivative(self.hidden_layer_output)

        self.weights_hidden_output += self.hidden_layer_output.T.dot(delta_output) * learning_rate
        self.bias_output += np.sum(delta_output, axis=0, keepdims=True) * learning_rate
        self.weights_input_hidden += self.inputs.T.dot(delta_hidden) * learning_rate
        self.bias_hidden += np.sum(delta_hidden, axis=0, keepdims=True) * learning_rate


if __name__ == "__main__":
    input_size = 4
    hidden_size = 8
    output_size = 1

    nn = NeuralNetwork(input_size, hidden_size, output_size)

    training_data = np.array([[0, 0, 1, 1], [0, 1, 1, 1], [1, 0, 1, 0], [1, 1, 1, 0]])
    target_data = np.array([[0], [1], [1], [0]])

    for _ in range(20000):
        nn.forward(training_data)
        nn.backward(target_data, learning_rate=0.001)

    new_data = np.array([0, 1, 0, 0])
    predicted_output = nn.forward(new_data)
    print(f"Predicted output: {predicted_output[0]}")
```