```python
import numpy as np


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


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


def train_xor_nn(epochs=100_000, learning_rate=0.1):
    # XOR input dataset
    X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
    # XOR output dataset
    y = np.array([[0], [1], [1], [0]])

    # Initialize weights and biases randomly
    np.random.seed(0)
    input_size, hidden_size, output_size = 2, 2, 1
    W1 = np.random.uniform(-1, 1, (input_size, hidden_size))
    b1 = np.random.uniform(-1, 1, (1, hidden_size))
    W2 = np.random.uniform(-1, 1, (hidden_size, output_size))
    b2 = np.random.uniform(-1, 1, (1, output_size))

    for epoch in range(epochs):
        # Forward pass
        hidden_input = np.dot(X, W1) + b1
        hidden_output = sigmoid(hidden_input)
        final_input = np.dot(hidden_output, W2) + b2
        final_output = sigmoid(final_input)

        # Compute error
        error = y - final_output

        # Backpropagation
        d_output = error * sigmoid_derivative(final_output)
        d_hidden = np.dot(d_output, W2.T) * sigmoid_derivative(hidden_output)

        # Update weights and biases
        W2 += np.dot(hidden_output.T, d_output) * learning_rate
        b2 += np.sum(d_output, axis=0, keepdims=True) * learning_rate
        W1 += np.dot(X.T, d_hidden) * learning_rate
        b1 += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

        if epoch % 1000 == 0:
            loss = np.mean(np.abs(error))
            print(f"Epoch {epoch}, Loss: {loss}")

    return W1, b1, W2, b2


def predict(X, W1, b1, W2, b2):
    hidden_output = sigmoid(np.dot(X, W1) + b1)
    final_output = sigmoid(np.dot(hidden_output, W2) + b2)
    return np.round(final_output)


# Train the neural network
W1, b1, W2, b2 = train_xor_nn()
```

```python

# Test the trained model
X_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_pred = predict(X_test, W1, b1, W2, b2)
print("Predictions:", y_pred.flatten())
print("Exact Predictions:", y_pred)

```