```python
import numpy as np

def sigmoid(a):
    """Sigmoid activation."""
    return 1.0 / (1.0 + np.exp(-a))

def sigmoid_derivative(y):
    """Derivative of sigmoid from output value."""
    return y * (1.0 - y)

class FFBPNetwork:
    """
    FFBP
    2 inputs -> 2 hidden nodes -> 1 output node
    """

    def __init__(self, w_input_hidden, w_hidden_output, bias_hidden, bias_output, eta):
        """
        Initialize network with weights and biases.

        Args:
            w_input_hidden: 2x2 array for input->hidden weights
            w_hidden_output: 2-element array for hidden->output weights
            bias_hidden: 2-element array for hidden layer biases
            bias_output: scalar bias for output node
            eta: learning rate
        """
        self.W_ih = np.array(w_input_hidden, dtype=float)
        self.W_ho = np.array(w_hidden_output, dtype=float)
        self.b_h = np.array(bias_hidden, dtype=float)
        self.b_o = float(bias_output)
        self.eta = float(eta)

    def forward(self, x):
        """
        Forward pass through the network.

        Returns: (A_h, y_h, A_o, y_o)
        """
        self.x = np.array(x, dtype=float)

        # Hidden lyer
        self.A_h = self.x @ self.W_ih + self.b_h
        self.y_h = sigmoid(self.A_h)

        # Output layer
        self.A_o = float(self.y_h @ self.W_ho + self.b_o)
        self.y_o = sigmoid(self.A_o)

        return self.A_h, self.y_h, self.A_o, self.y_o

    def backward(self, x, d):
        """
        Backward pass - compute gradients and update weights.

        Args:
            x: input vector
            d: desired output
        """
        # Forward pass
        self.forward(x)

        # Output layer gradients
        e = d - self.y_o
        delta_o = e * (-1.0) * sigmoid_derivative(self.y_o)

        grad_W_ho = delta_o * self.y_h
        grad_b_o = delta_o * 1.0

        # Hidden layer gradients
        delta_h = delta_o * self.W_ho * sigmoid_derivative(self.y_h)
        grad_W_ih = np.outer(self.x, delta_h)
        grad_b_h = delta_h * 1.0

        # update weights and biases
        self.W_ho -= self.eta * grad_W_ho
        self.b_o -= self.eta * grad_b_o
        self.W_ih -= self.eta * grad_W_ih
        self.b_h -= self.eta * grad_b_h

    def compute_error(self, x, d):
        """Compute total error E = 0.5 * (d - y)^2"""
        self.forward(x)
        return 0.5 * (d - self.y_o) ** 2

    def get_output(self, x):
        """Get network output for given input."""
        self.forward(x)
        return self.y_o


def method_1(w_ih_init, w_ho_init, b_h_init, b_o_init, eta, cycles=15):
    """
    Method 1: Alternate between the two input/output pairs for each cycle.
    """
    net = FFBPNetwork(w_ih_init, w_ho_init, b_h_init, b_o_init, eta)

    pair1_input = np.array([1.0, 1.0])
    pair1_output = 0.9
    pair2_input = np.array([-1.0, -1.0])
    pair2_output = 0.05

    for cycle in range(cycles):
        # Train on first pair
        net.backward(pair1_input, pair1_output)
        # Train on second pair
        net.backward(pair2_input, pair2_output)

    # After training, compute outputs and errors
    output1 = net.get_output(pair1_input)
    error1 = net.compute_error(pair1_input, pair1_output)

    output2 = net.get_output(pair2_input)
    error2 = net.compute_error(pair2_input, pair2_output)

    return output1, error1, output2, error2, net


def method_2(w_ih_init, w_ho_init, b_h_init, b_o_init, eta, iterations=15):
    """
    Method 2: Train on first pair for 15 iter, then second pair for 15 iter.
    """
    net = FFBPNetwork(w_ih_init, w_ho_init, b_h_init, b_o_init, eta)
```

```
    pair1_input = np.array([1.0, 1.0])
    pair1_output = 0.9
    pair2_input = np.array([-1.0, -1.0])
    pair2_output = 0.05

    # train on first pair for 15 iter
    for _ in range(iterations):
        net.backward(pair1_input, pair1_output)

    # Train on second pair for 15 iter
    for _ in range(iterations):
        net.backward(pair2_input, pair2_output)

    # After training, compute outputs and errors for both pairs
    output1 = net.get_output(pair1_input)
    error1 = net.compute_error(pair1_input, pair1_output)

    output2 = net.get_output(pair2_input)
    error2 = net.compute_error(pair2_input, pair2_output)

    return output1, error1, output2, error2, net


if __name__ == "__main__":
    # Initial parameters
    w_ih_init = [[0.3, 0.3],
                 [0.3, 0.3]]
    w_ho_init = [0.8, 0.8]
    b_h_init = [0.0, 0.0]
    b_o_init = 0.0
    eta = 1.0


    print("METHOD 1: Alternating training\n")


    output1_m1, error1_m1, output2_m1, error2_m1, net_m1 = method_1(
        w_ih_init, w_ho_init, b_h_init, b_o_init, eta, cycles=15
    )

    print(f"\nAfter 15 cycles of alternating training:")
    print(f"\nInput [1.0, 1.0] (desired: 0.9):")
    print(f"  Output: {output1_m1:.4f}")
    print(f"  Big E:  {error1_m1:.4f}")

    print(f"\nInput [-1.0, -1.0] (desired: 0.05):")
    print(f"  Output: {output2_m1:.4f}")
    print(f"  Big E:  {error2_m1:.4f}")

    print("\n")
    print("METHOD 2: Sequential training")

    output1_m2, error1_m2, output2_m2, error2_m2, net_m2 = method_2(
        w_ih_init, w_ho_init, b_h_init, b_o_init, eta, iterations=15
    )

    print(f"\nAfter 15 iterations on each pair:")
    print(f"\nInput [1.0, 1.0] (desired: 0.9):")
    print(f"  Output: {output1_m2:.4f}")
    print(f"  Big E:  {error1_m2:.4f}")

    print(f"\nInput [-1.0, -1.0] (desired: 0.05):")
    print(f"  Output: {output2_m2:.4f}")
    print(f"  Big E:  {error2_m2:.4f}")

    print("\n")
    print("ANSWERS FOR SUBMISSION")
    print("\n")
    print(f"Q1 (Method 1, input [1,1], output):     {output1_m1:.4f}")
    print(f"Q2 (Method 1, input [1,1], Big E):      {error1_m1:.4f}")
    print(f"Q3 (Method 1, input [-1,-1], output):   {output2_m1:.4f}")
    print(f"Q4 (Method 1, input [-1,-1], Big E):    {error2_m1:.4f}")
    print(f"Q5 (Method 2, input [1,1], output):     {output1_m2:.4f}")
    print(f"Q6 (Method 2, input [1,1], Big E):      {error1_m2:.4f}")
    print(f"Q7 (Method 2, input [-1,-1], output):   {output2_m2:.4f}")
    print(f"Q8 (Method 2, input [-1,-1], Big E):    {error2_m2:.4f}")
```