```python
import numpy as np

class Perceptron:
    """
    A simple Perceptron classifier.

    Parameters
    ----------
    learning_rate : float
        The learning rate (between 0.0 and 1.0).
    n_iters : int
        The number of passes over the training dataset (epochs).

    Attributes
    ----------
    weights : 1d-array
        Weights after fitting.
    bias : scalar
        Bias unit after fitting.
    """

    def __init__(self, learning_rate=0.1, n_iters=100):
        self.learning_rate = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = 0

    def _unit_step_func(self, x):
        """Activation function"""
        return np.where(x >= 0, 1, 0)

    def fit(self, X, y):
        """
        Fit training data.

        Parameters
        ----------
        X : {array-like}, shape = [n_samples, n_features]
            Training vectors.
        y : array-like, shape = [n_samples]
            Target values.
        """
        n_samples, n_features = X.shape

        # Initialize weights
        self.weights = np.zeros(n_features)

        # Training loop
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                # Calculate linear output: w.x + b
                linear_output = np.dot(x_i, self.weights) + self.bias

                # Apply activation function
                y_predicted = self._unit_step_func(linear_output)

                # Perceptron update rule
                # update = learning_rate * (target - prediction)
                update = self.learning_rate * (y[idx] - y_predicted)

                # Update weights and bias
                self.weights += update * x_i
                self.bias += update

    def predict(self, X):
        """Return class label after unit step"""
        linear_output = np.dot(X, self.weights) + self.bias
        return self._unit_step_func(linear_output)

# --- Common Inputs ---
# Inputs for 2-bit logic gates
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])

# --- 1. AND Gate ---
y_and = np.array([0, 0, 0, 1])

p_and = Perceptron(learning_rate=0.1, n_iters=10)
p_and.fit(X, y_and)

print("--- AND Gate ---")
print(f"Learned Weights: {p_and.weights}")
print(f"Learned Bias: {p_and.bias}\n")
print("Truth Table (Predicted):")
for x_input in X:
    prediction = p_and.predict(x_input)
    print(f"{x_input[0]} AND {x_input[1]} = {prediction}")

# --- 2. OR Gate ---
y_or = np.array([0, 1, 1, 1])

p_or = Perceptron(learning_rate=0.1, n_iters=10)
p_or.fit(X, y_or)

print("\n--- OR Gate ---")
print(f"Learned Weights: {p_or.weights}")
print(f"Learned Bias: {p_or.bias}\n")
print("Truth Table (Predicted):")
for x_input in X:
    prediction = p_or.predict(x_input)
    print(f"{x_input[0]} OR {x_input[1]} = {prediction}")

# --- 3. NOT Gate ---
# (Note: NOT gate only has one input)
X_not = np.array([[0], [1]])
y_not = np.array([1, 0])

p_not = Perceptron(learning_rate=0.1, n_iters=10)
p_not.fit(X_not, y_not)

print("\n--- NOT Gate ---")
print(f"Learned Weights: {p_not.weights}")
print(f"Learned Bias: {p_not.bias}\n")
print("Truth Table (Predicted):")
for x_input in X_not:
    prediction = p_not.predict(x_input)
    print(f"NOT {x_input[0]} = {prediction}")

# --- 4. XOR Gate (Demonstrating Limitation) ---
# A single-layer perceptron CANNOT solve XOR
y_xor = np.array([0, 1, 1, 0])

p_xor = Perceptron(learning_rate=0.1, n_iters=100) # More iterations
p_xor.fit(X, y_xor)

print("\n--- XOR Gate (Limitation) ---")
print(f"Learned Weights (after 100 iterations): {p_xor.weights}")
print(f"Learned Bias (after 100 iterations): {p_xor.bias}\n")
print("Truth Table (Predicted):")
print("Notice how the model fails to learn the correct pattern:")
for x_input in X:
    prediction = p_xor.predict(x_input)
    print(f"{x_input[0]} XOR {x_input[1]} = {prediction} (Expected:
{y_xor[X.tolist().index(x_input.tolist())]})")
```

```
--- AND Gate ---
Learned Weights: [0.2 0.1]
Learned Bias: -0.20000000000000004

Truth Table (Predicted):
0 AND 0 = 0
0 AND 1 = 0
1 AND 0 = 0
1 AND 1 = 1

--- OR Gate ---
```

```
Learned Weights: [0.1 0.1]
Learned Bias: -0.1

Truth Table (Predicted):
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

--- NOT Gate ---
Learned Weights: [-0.1]
Learned Bias: 0.0

Truth Table (Predicted):
NOT 0 = 1
NOT 1 = 0

--- XOR Gate (Limitation) ---
Learned Weights (after 100 iterations): [-0.1  0. ]
Learned Bias (after 100 iterations): 0.0

Truth Table (Predicted):
Notice how the model fails to learn the correct pattern:
0 XOR 0 = 1 (Expected: 0)
0 XOR 1 = 1 (Expected: 1)
1 XOR 0 = 0 (Expected: 1)
1 XOR 1 = 0 (Expected: 0)
```