```
import numpy as np

# XOR dataset
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])

# XOR outputs (truth table)
y = np.array([0, 1, 1, 0])



class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=10):
        self.weights = np.zeros(input_size + 1)  # Including bias
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation(self, x):
        return 1 if x >= 0 else 0  # Step function (MCP activation)

    def predict(self, x):
        weighted_sum = np.dot(x, self.weights[1:]) + self.weights[0]  # w.X + bias
        return self.activation(weighted_sum)

    def train(self, X, y):
        for epoch in range(self.epochs):
            total_error = 0
            for i in range(len(X)):
                prediction = self.predict(X[i])
                error = y[i] - prediction
                self.weights[1:] += self.learning_rate * error * X[i]
                self.weights[0] += self.learning_rate * error  # Bias update
                total_error += abs(error)
            print(f'Epoch {epoch+1}/{self.epochs}, Total Error: {total_error}')
            if total_error == 0:
                break
        print(f'Final Weights: {self.weights}')

# Train perceptron on XOR data
p = Perceptron(input_size=2, learning_rate=0.1, epochs=10)
p.train(X, y)

# Test perceptron
for x in X:
    print(f'Input: {x}, Predicted Output: {p.predict(x)}')
```

```
Epoch 1/10, Total Error: 3
Epoch 2/10, Total Error: 3
Epoch 3/10, Total Error: 4
Epoch 4/10, Total Error: 4
Epoch 5/10, Total Error: 4
Epoch 6/10, Total Error: 4
Epoch 7/10, Total Error: 4
Epoch 8/10, Total Error: 4
Epoch 9/10, Total Error: 4
Epoch 10/10, Total Error: 4
Final Weights: [ 0.  -0.1  0. ]
Input: [0 0], Predicted Output: 1
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 0
Input: [1 1], Predicted Output: 0
```

the perceptron fails to classify the XOR gate correctly. This is because the XOR gate is not linearly separable, and a single-layer perceptron can only solve linearly separable problems.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

# Create MLP model
model = Sequential()
model.add(Input(shape=(2,)))  # Define the input layer with shape
model.add(Dense(2, activation='relu'))  # Hidden layer with 2 neurons
model.add(Dense(1, activation='sigmoid'))  # Output layer with 1 neuron

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model (increase epochs to improve performance)
```

```
history = model.fit(X, y, epochs=500, verbose=0)

# Evaluate the model
_, accuracy = model.evaluate(X, y)
print(f'Model Accuracy: {accuracy * 100:.2f}%')

# Predict the outputs
predictions = model.predict(X)
predictions = [1 if p > 0.5 else 0 for p in predictions]
for i in range(len(X)):
    print(f'Input: {X[i]}, Predicted Output: {predictions[i]}')
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 159ms/step - accuracy: 1.0000 - loss: 0.5757
Model Accuracy: 100.00%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
Input: [0 0], Predicted Output: 0
Input: [0 1], Predicted Output: 1
Input: [1 0], Predicted Output: 1
Input: [1 1], Predicted Output: 0
```

Single Layer Perceptron: The single-layer perceptron fails to solve the XOR gate problem because it cannot draw a linear boundary to classify non-linearly separable data.

Multi-Layer Perceptron (MLP): The MLP can successfully classify the XOR gate's outputs because it introduces a hidden layer with non-linear activation functions (ReLU). This allows it to learn non-linear decision boundaries.

A Single Layer Perceptron is insufficient for solving the XOR problem due to the linear separability constraint. A Multi-Layer Perceptron successfully classifies XOR by introducing non-linearities via hidden layers.