```python
import numpy as np
from sklearn.datasets import make_classification
import matplotlib.pyplot as plt

# 1. LINEARLY SEPARABLE DATASET
X_linear, y_linear = make_classification(
    n_samples=500, n_features=2, n_informative=2, n_redundant=0,
    n_clusters_per_class=1, class_sep=2.0, random_state=42
)
y_linear = 2 * y_linear - 1
print(f"Linear dataset: {X_linear.shape}, classes: {np.unique(y_linear)}")

X_xor = np.array([[0,0], [0,1], [1,0], [1,1]])
y_xor = np.array([0,1,1,0])
print(f"XOR dataset: {X_xor.shape}, classes: {np.unique(y_xor)}")
```

```
Linear dataset: (500, 2), classes: [-1  1]
XOR dataset: (4, 2), classes: [0 1]
```

```python
def perceptron_train(X, y, lr=0.01, epochs=1000):
    """Single-layer perceptron for linear separation"""
    weights = np.random.randn(X.shape[1]) * 0.01
    bias = 0
    for epoch in range(epochs):
        linear_out = X @ weights + bias
        predictions = np.heaviside(linear_out, 1)

        weights += lr * (y - predictions) @ X
        bias += lr * np.mean(y - predictions)
    return weights, bias

def perceptron_predict(X, weights, bias):
    linear_out = X @ weights + bias
    return np.heaviside(linear_out, 1)


w_lin, b_lin = perceptron_train(X_linear, y_linear)
pred_lin = perceptron_predict(X_linear, w_lin, b_lin)
acc_lin = np.mean(pred_lin == y_linear)
print(f"Linear dataset accuracy: {acc_lin:.1%}")


w_xor, b_xor = perceptron_train(X_xor, y_xor, epochs=5000, lr=0.1)
pred_xor = perceptron_predict(X_xor, w_xor, b_xor)
acc_xor = np.mean(pred_xor == y_xor)
print(f"XOR dataset accuracy: {acc_xor:.1%}")
```

```
Linear dataset accuracy: 47.4%
XOR dataset accuracy: 50.0%
```

```python
def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -250, 250)))

def relu(z):
    return np.maximum(0, z)

def relu_deriv(z):
    return (z > 0).astype(float)

class SimpleMLP:
    def __init__(self, input_size=2, hidden_size=8, output_size=1):

        self.W1 = np.random.randn(input_size, hidden_size) * 0.1
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * 0.1
        self.b2 = np.zeros((1, output_size))

    def forward(self, X):
        self.z1 = X @ self.W1 + self.b1
        self.a1 = relu(self.z1)
        self.z2 = self.a1 @ self.W2 + self.b2
        self.a2 = sigmoid(self.z2)
        return self.a2

    def backward(self, X, y_target, lr=1.0):
        m = X.shape[0]


        dz2 = self.a2 - y_target
        dW2 = (1/m) * (self.a1.T @ dz2)
        db2 = (1/m) * np.sum(dz2, axis=0, keepdims=True)


        da1 = dz2 @ self.W2.T
        dz1 = da1 * relu_deriv(self.z1)
        dW1 = (1/m) * (X.T @ dz1)
        db1 = (1/m) * np.sum(dz1, axis=0, keepdims=True)


        self.W2 -= lr * dW2
        self.b2 -= lr * db2
        self.W1 -= lr * dW1
        self.b1 -= lr * db1

    def train(self, X, y, epochs=10000):
        if y.ndim == 1:
            y_reshaped = y.reshape(-1, 1)
        else:
            y_reshaped = y

        for epoch in range(epochs):
            self.forward(X)
            self.backward(X, y_reshaped)
            if epoch % 2000 == 0:
                loss = np.mean((self.forward(X) - y_reshaped)**2)
                print(f"Epoch {epoch}, Loss: {loss:.4f}")

    def predict(self, X):
        return (self.forward(X) > 0.5).astype(int).flatten()


mlp_xor = SimpleMLP()
mlp_xor.train(X_xor, y_xor)
pred_xor_mlp = mlp_xor.predict(X_xor)
acc_xor_mlp = np.mean(pred_xor_mlp == y_xor)
print(f"\nXOR dataset accuracy: {acc_xor_mlp:.1%}")

y_linear_01 = (y_linear + 1) / 2
mlp_lin = SimpleMLP()
mlp_lin.train(X_linear, y_linear_01)
pred_lin_mlp = mlp_lin.predict(X_linear)
acc_lin_mlp = np.mean(pred_lin_mlp == y_linear_01)
print(f"Linear dataset accuracy: {acc_lin_mlp:.1%}")
```

```
Epoch 0, Loss: 0.2487
Epoch 2000, Loss: 0.0000
Epoch 4000, Loss: 0.0000
Epoch 6000, Loss: 0.0000
Epoch 8000, Loss: 0.0000

XOR dataset accuracy: 100.0%
Epoch 0, Loss: 0.2377
Epoch 0, Loss: 0.2377
Epoch 2000, Loss: 0.0083
Epoch 4000, Loss: 0.0031
Epoch 6000, Loss: 0.0023
Epoch 8000, Loss: 0.0021
Linear dataset accuracy: 99.8%
```

```
Start coding or generate with AI.
```