LAB 4 > dltlab6.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split


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

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

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

def init_network(input_size, hidden1, hidden2, output_size):
    np.random.seed(42)
    weights = {
        "W1": np.random.randn(input_size, hidden1) / np.sqrt(input_size),
        "b1": np.zeros((1, hidden1)),
        "W2": np.random.randn(hidden1, hidden2) / np.sqrt(hidden1),
        "b2": np.zeros((1, hidden2)),
        "W3": np.random.randn(hidden2, output_size) / np.sqrt(hidden2),
        "b3": np.zeros((1, output_size)),
    }
    return weights


def forward(X, weights):
    Z1 = X.dot(weights["W1"]) + weights["b1"]
    A1 = relu(Z1)

    Z2 = A1.dot(weights["W2"]) + weights["b2"]
    A2 = relu(Z2)

    Z3 = A2.dot(weights["W3"]) + weights["b3"]
    A3 = sigmoid(Z3)

    cache = (X, Z1, A1, Z2, A2, Z3, A3)
```

LAB 4 > dltlab6.py > ...

```
29  def forward(X, weights):
39      cache = (X, Z1, A1, Z2, A2, Z3, A3)
40      return A3, cache
41
42  def backward(y, weights, cache, learning_rate=0.01):
43      X, Z1, A1, Z2, A2, Z3, A3 = cache
44      m = y.shape[0]
45
46      # output layer
47      dZ3 = A3 - y
48      dW3 = (A2.T @ dZ3) / m
49      db3 = np.sum(dZ3, axis=0, keepdims=True) / m
50
51      # hidden layer 2
52      dA2 = dZ3.dot(weights["W3"].T)
53      dZ2 = dA2 * relu_deriv(Z2)
54      dW2 = (A1.T @ dZ2) / m
55      db2 = np.sum(dZ2, axis=0, keepdims=True) / m
56
57      # hidden layer 1
58      dA1 = dZ2.dot(weights["W2"].T)
59      dZ1 = dA1 * relu_deriv(Z1)
60      dW1 = (X.T @ dZ1) / m
61      db1 = np.sum(dZ1, axis=0, keepdims=True) / m
62
63      # update params
64      weights["W1"] -= learning_rate * dW1
65      weights["b1"] -= learning_rate * db1
66      weights["W2"] -= learning_rate * dW2
67      weights["b2"] -= learning_rate * db2
68      weights["W3"] -= learning_rate * dW3
69      weights["b3"] -= learning_rate * db3
70
71      return weights
72
73  def train(X, y, hidden1=32, hidden2=16, epochs=2000, lr=0.01):
74      weights = init_network(X.shape[1], hidden1, hidden2, y.shape[1])
75
76      # store history for plotting
```

LAB 4 > dltlab6.py > ...

```python
def train(X, y, hidden1=32, hidden2=16, epochs=2000, lr=0.01):
    # store history for plotting
    loss_history = []
    acc_history = []

    for epoch in range(epochs):
        y_pred, cache = forward(X, weights)

        # binary cross-entropy loss
        loss = -np.mean(y * np.log(y_pred+1e-9) + (1-y)*np.log(1-y_pred+1e-9))

        weights = backward(y, weights, cache, learning_rate=lr)

        # compute accuracy
        preds = (y_pred > 0.5).astype(int)
        acc = np.mean(preds == y)

        # save for curve
        loss_history.append(loss)
        acc_history.append(acc)

        if epoch % 200 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}, Accuracy: {acc:.4f}")

    return weights, loss_history, acc_history

def evaluate(X, y, weights):
    y_pred, _ = forward(X, weights)
    preds = (y_pred > 0.5).astype(int)
    acc = np.mean(preds == y)
    return acc

def plot_curves(loss_history, acc_history):
    epochs = range(len(loss_history))

    plt.figure(figsize=(12,5))

    # Loss curve
    plt.subplot(1,2,1)
```

LAB 4 > dltlab6.py > ...

```python
def plot_curves(loss_history, acc_history):
    epochs = range(len(loss_history))

    plt.figure(figsize=(12,5))

    # Loss curve
    plt.subplot(1,2,1)
    plt.plot(epochs, loss_history, label="Loss", color="red")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.title("Training Loss Curve")
    plt.legend()

    # Accuracy curve
    plt.subplot(1,2,2)
    plt.plot(epochs, acc_history, label="Accuracy", color="blue")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Training Accuracy Curve")
    plt.legend()

    plt.show()


if __name__ == "__main__":
    # generate data
    X, y = make_moons(n_samples=500, noise=0.2, random_state=42)
    y = y.reshape(-1, 1)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # train
    trained_weights, loss_hist, acc_hist = train(X_train, y_train, hidden1=32, hidden2=16, epochs=2000, lr=0.01)

    # evaluate
    test_acc = evaluate(X_test, y_test, trained_weights)
    print(f"\nFinal Test Accuracy: {test_acc:.4f}")

    # plot training curves
```