```
import numpy as np
import pandas as pd

# Dataset
data = {
    'CGPA': [8.5, 7.2, 9.1, 6.8, 7.5, 8, 7.9, 8.3, 6.5, 9],
    '10th Score': [85, 78, 90, 70, 75, 80, 79, 83, 65, 92],
    '12th Score': [88, 74, 92, 65, 78, 81, 77, 85, 60, 95],
    'IQ': [120, 110, 130, 105, 115, 118, 113, 125, 100, 135],
    'Placement': [1, 0, 1, 0, 0, 1, 1, 1, 0, 1]
}
df = pd.DataFrame(data)

# Feature and label extraction
X = df[['CGPA', '10th Score', '12th Score', 'IQ']].values.T
Y = df[['Placement']].values.T

# Normalization
X = X / np.max(X, axis=1, keepdims=True)

# Initialize Parameters
def init_parameters(layer_dims):
    np.random.seed(42)
    parameters = {}
    L = len(layer_dims)
    for i in range(1, L):
        parameters['w'+str(i)] = np.random.randn(layer_dims[i-1], layer_dims[i])
        parameters['b'+str(i)] = np.zeros((layer_dims[i], 1))
    return parameters

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

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

def relu_backward(dA, Z):
    dZ = np.array(dA, copy=True)
    dZ[Z <= 0] = 0
    return dZ

def sigmoid_backward(AL, Y):
    return AL - Y

# Forward
def linear_forward(A_prev, W, B):
    return np.dot(W.T, A_prev) + B

def L_layer_forward(X, parameters):
    A = X
    caches = []
    L = len(parameters) // 2
```

```
    for i in range(1, L):
        A_prev = A
        W = parameters['w' + str(i)]
        B = parameters['b' + str(i)]
        Z = linear_forward(A_prev, W, B)
        A = relu(Z)
        caches.append((A_prev, W, B, Z))

    # Output layer
    W_out = parameters['w' + str(L)]
    B_out = parameters['b' + str(L)]
    Z_out = linear_forward(A, W_out, B_out)
    AL = sigmoid(Z_out)
    caches.append((A, W_out, B_out, Z_out))
    return AL, caches

# Backward
def L_layer_backward(AL, Y, caches):
    grads = {}
    L = len(caches)
    m = AL.shape[1]
    dAL = sigmoid_backward(AL, Y)

    A_prev, W, B, Z = caches[-1]
    grads['dw' + str(L)] = np.dot(A_prev, dAL.T) / m
    grads['db' + str(L)] = np.sum(dAL, axis=1, keepdims=True) / m
    dA_prev = np.dot(W, dAL)

    for l in reversed(range(L - 1)):
        A_prev, W, B, Z = caches[l]
        dZ = relu_backward(dA_prev, Z)
        grads['dw' + str(l+1)] = np.dot(A_prev, dZ.T) / m
        grads['db' + str(l+1)] = np.sum(dZ, axis=1, keepdims=True) / m
        dA_prev = np.dot(W, dZ)

    return grads

# Update
def update_parameters(parameters, grads, learning_rate):
    L = len(parameters) // 2
    for l in range(1, L + 1):
        parameters['w' + str(l)] -= learning_rate * grads['dw' + str(l)]
        parameters['b' + str(l)] -= learning_rate * grads['db' + str(l)]
    return parameters

# Train
def train_model(X, Y, layer_dims, learning_rate=0.1, epochs=5000):
    parameters = init_parameters(layer_dims)
    for i in range(epochs):
        AL, caches = L_layer_forward(X, parameters)
        grads = L_layer_backward(AL, Y, caches)
        parameters = update_parameters(parameters, grads, learning_rate)
        if i % 1000 == 0:
            loss = -np.mean(Y * np.log(AL + 1e-8) + (1 - Y) * np.log(1 - AL + 1e-
            print(f"Epoch {i}: Loss = {loss:.4f}")
```

```
    return parameters

# Run training
params = train_model(X, Y, [4, 3, 1], learning_rate=0.1, epochs=5000)

# Predict
y_pred, _ = L_layer_forward(X, params)
predictions = (y_pred > 0.5).astype(int)

# Result
print("\nPredictions:\n", predictions)
print("Actual:\n", Y)
```

```
Epoch 0: Loss = 0.6919
Epoch 1000: Loss = 0.3094
Epoch 2000: Loss = 0.1546
Epoch 3000: Loss = 0.0948
Epoch 4000: Loss = 0.0609

Predictions:
 [[1 0 1 0 0 1 1 1 0 1]]
Actual:
 [[1 0 1 0 0 1 1 1 0 1]]
```

Start coding or generate with AI.