Experiment 6: Implement Backpropagation in a Simple MLP

Aim: Implement Backpropagation in a Simple MLP

Objective:

1. To understand and implement the backpropagation algorithm in a simple Multilayer Perceptron (MLP).

2. To study the error calculation and weight update process.

3. To visualize the reduction of error over training epochs.

Code:

```
import numpy as np
import pandas as pd
from google.colab import drive

drive.mount('/content/drive')
!ls "/content/drive/My Drive/ANN/Student_dataset.xlsx"

# Initialize parameters
def init_parameters(layer_dimension):
    np.random.seed(42)
    parameters = {}
    L = len(layer_dimension)
    for i in range(1, L):
        parameters['w'+ str(i)] = np.random.randn(layer_dimension[i-1],
layer_dimension[i]) * 0.1
        parameters['b'+ str(i)] = np.zeros((layer_dimension[i], 1))
    return parameters

# Activation functions and their derivatives
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
```

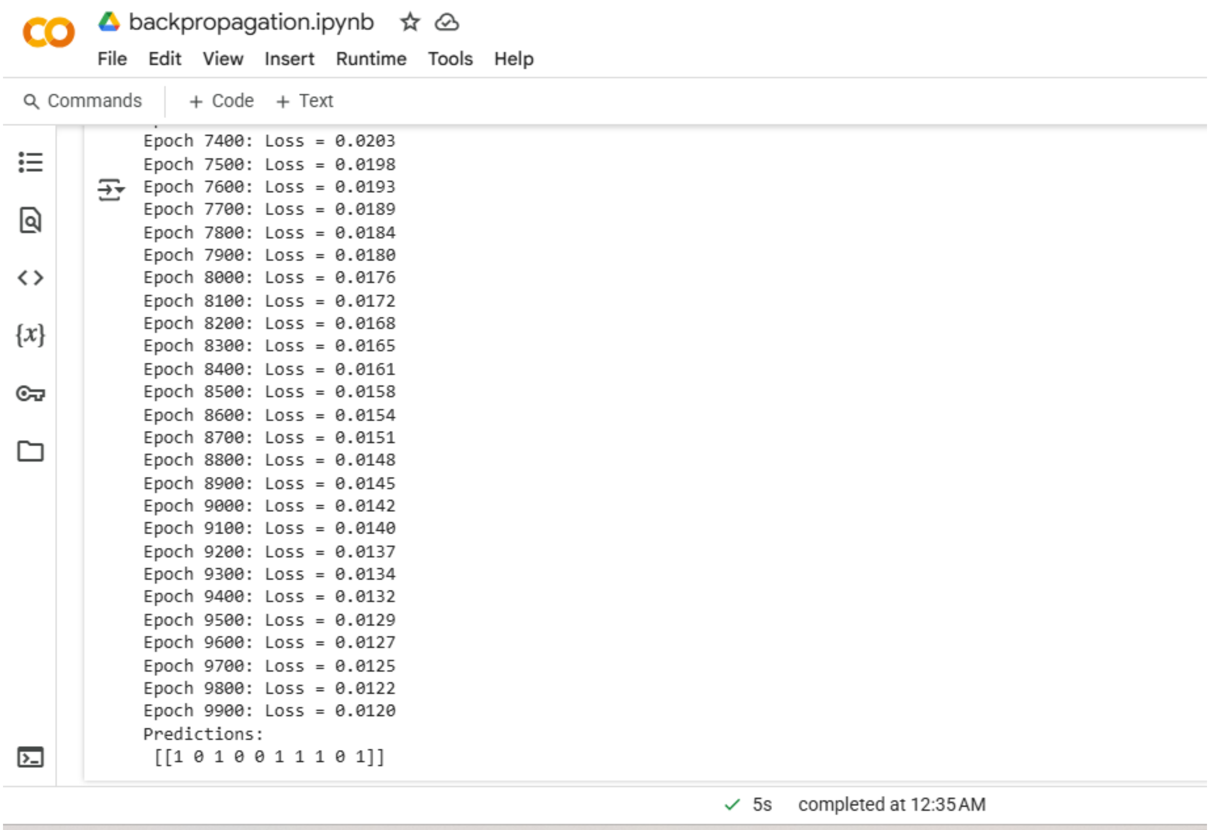
```
# Linear forward
def linear_forward(A_prev, W, B):
    z = np.dot(W.T, A_prev) + B
    return z

# Forward propagation
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)
        cache = (A_prev, W, B, Z)
        caches.append(cache)
    # 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)
    cache = (A, W_out, B_out, Z_out)
    caches.append(cache)
    return AL, caches

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

    # Output layer gradients
    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
```

```
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

def train_model(X, Y, layer_dims, learning_rate=0.01, epochs=1000):
    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 % 100 == 0:
            loss = -np.mean(Y * np.log(AL + 1e-8) + (1 - Y) * np.log(1
- AL + 1e-8))
            print(f"Epoch {i}: Loss = {loss:.4f}")
    return parameters


# Load data
file_path = "/content/drive/My Drive/ANN/Student_dataset.xlsx"
df = pd.read_excel(file_path)
print(f"Dataframe = \n{df}")

X = df[['CGPA', '10th Score', '12th Score', 'IQ']].values.T
Y = df[['Placement']].values.T
X = X / np.max(X, axis=1, keepdims=True)

# # Initialize and run
# parameters = init_parameters([4, 2, 1])
# y_cap, caches = L_layer_forward(X, parameters)
# print("Output =", y_cap)

# # Run backward propagation
# grads = L_layer_backward(y_cap, Y, caches)
# print("Gradients =", grads)

# Train the model
parameters = train_model(X, Y, [4, 2, 1], learning_rate=0.1,
epochs=10000)
```

```
# Predict
y_test, _ = L_layer_forward(X, parameters)
predictions = (y_test > 0.5).astype(int)
print("Predictions:\n", predictions)
```

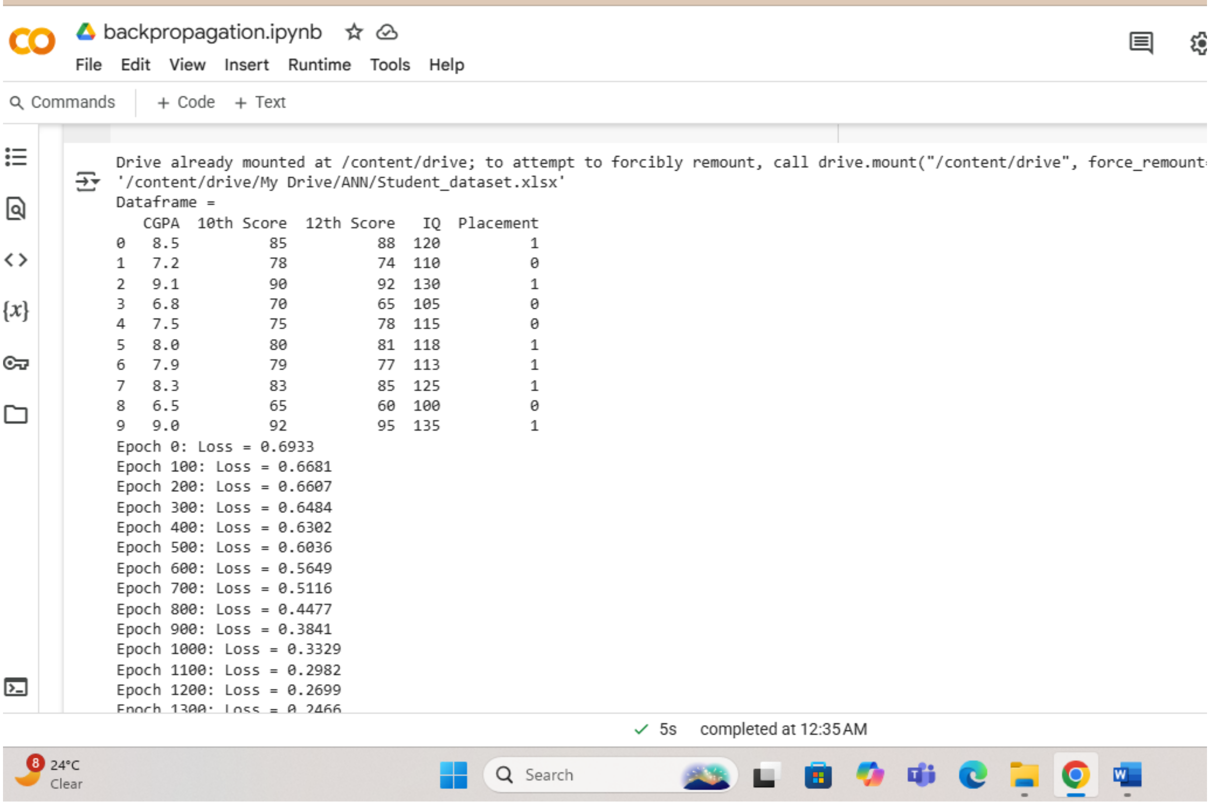
Output:

Test:



```
Epoch 7400: Loss = 0.0203
Epoch 7500: Loss = 0.0198
Epoch 7600: Loss = 0.0193
Epoch 7700: Loss = 0.0189
Epoch 7800: Loss = 0.0184
Epoch 7900: Loss = 0.0180
Epoch 8000: Loss = 0.0176
Epoch 8100: Loss = 0.0172
Epoch 8200: Loss = 0.0168
Epoch 8300: Loss = 0.0165
Epoch 8400: Loss = 0.0161
Epoch 8500: Loss = 0.0158
Epoch 8600: Loss = 0.0154
Epoch 8700: Loss = 0.0151
Epoch 8800: Loss = 0.0148
Epoch 8900: Loss = 0.0145
Epoch 9000: Loss = 0.0142
Epoch 9100: Loss = 0.0140
Epoch 9200: Loss = 0.0137
Epoch 9300: Loss = 0.0134
Epoch 9400: Loss = 0.0132
Epoch 9500: Loss = 0.0129
Epoch 9600: Loss = 0.0127
Epoch 9700: Loss = 0.0125
Epoch 9800: Loss = 0.0122
Epoch 9900: Loss = 0.0120
Predictions:
 [[1 0 1 0 0 1 1 1 0 1]]
```

Train:



```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
'/content/drive/My Drive/ANN/Student_dataset.xlsx'
Dataframe =
   CGPA  10th Score  12th Score   IQ  Placement
0   8.5          85          88  120          1
1   7.2          78          74  110          0
2   9.1          90          92  130          1
3   6.8          70          65  105          0
4   7.5          75          78  115          0
5   8.0          80          81  118          1
6   7.9          79          77  113          1
7   8.3          83          85  125          1
8   6.5          65          60  100          0
9   9.0          92          95  135          1
Epoch 0: Loss = 0.6933
Epoch 100: Loss = 0.6681
Epoch 200: Loss = 0.6607
Epoch 300: Loss = 0.6484
Epoch 400: Loss = 0.6302
Epoch 500: Loss = 0.6036
Epoch 600: Loss = 0.5649
Epoch 700: Loss = 0.5116
Epoch 800: Loss = 0.4477
Epoch 900: Loss = 0.3841
Epoch 1000: Loss = 0.3329
Epoch 1100: Loss = 0.2982
Epoch 1200: Loss = 0.2699
Epoch 1300: Loss = 0.2466
```