RA2311047010016

S.No.	Date	Title	Signature
1	07.08.2025	Exploring the Deep Learning platforms	21/8/25
2.	07.08.2025	Implement a classifier using open source dataset	
3.	14.08.2025	Study of classifiers with respect to Statistical parameters	14/8/25
4.	22.08.2025	To build and train a simple feed forward Network (~~FFNN~~) on MNIST dataset.	23/8/25
5.	22.8.25	Study different activation function used in NN	6/9/25
6.	9.9.25	Implement Gradient Descent & Backpropagation in ~~ANN~~	
7.	16.09.2025	Build a CNN model to classify Cat and dog image	23/9/25

9-9-25

6. Implement gradient descent and backpropagation in deep neural network (aim, objective, pseudocode).

Aim:

* To implement gradient descent and backpropagation in a deep neural network to optimize the weights and biases for minimizing the error between predicted and actual outputs.

Objective

* Understand the working of gradient descent in optimizing neural networks.

* Learn how backpropogation computes gradients efficiently.

* Implement a multi-layer (deep) neural network from stratch.

* Train the network on sample data using backpropagation and gradient descent.

* observe how the loss decreases over epochs.

Prediction :

tensor ([50.],
[1.]
[1.],
[0.7])

Ground truth :

Tensor ([[0.],
[1.],
[1.],
[0.]]

Formula used:

Linear $Z^{[l]} = W^{[l]} + A^{[l-1]} + b^{[l]}$

Activation (sigmoid, relu).

$$A^{[l]} = \sigma(Z^{[l]})$$

Loss function:

$$L = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-\hat{y}^{(i)})\log(1-\hat{y}^{(i)})\right].$$

pseudocode.

1. Initialize weights W[l] and biases b[l] for all layers.

2. Set learning rate and number of epochs.

3. For each epoch;

   a) Forward pass:

      - For each layer:

        Z [l] = W[l] * A [l-1] + b[l]

        A [l] = Activation (Z[l])

   b) Comput Loss using predicted output A [L] and actual output Y output

epoch 0 - Loss : 0.7198

epoch 500 - Loss : 0.6869

epoch 1000 - Loss : 0.6755

epoch 1500 - Loss : 0.6402

epoch 2000 - loss : 0.5343

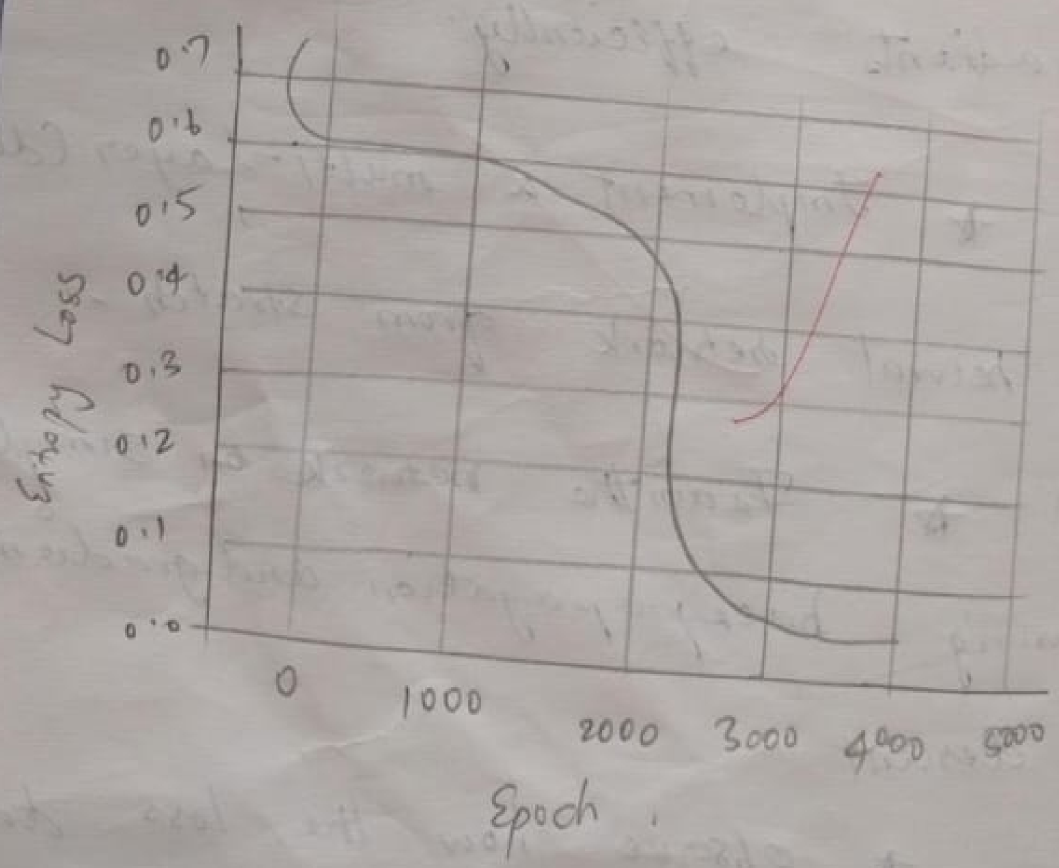
epoch 2500 - loss : 0.2709

epoch 3000 - Loss : 0.0881

epoch 3500 - loss : 0.0417

epoch : 4000 - loss : 0.0453

epoch 4500 - loss : 0.0177



Backward pass :

- Compute dA[L] from loss -
- For each layer in reverse:

dz[l] = dA [l] * Activation_ Derivative (z[l])

dw[l], db[l] = Gradients from dz [l]

dA [l-1] = W[l]^T . * dz [l]

d.) Update parameters :

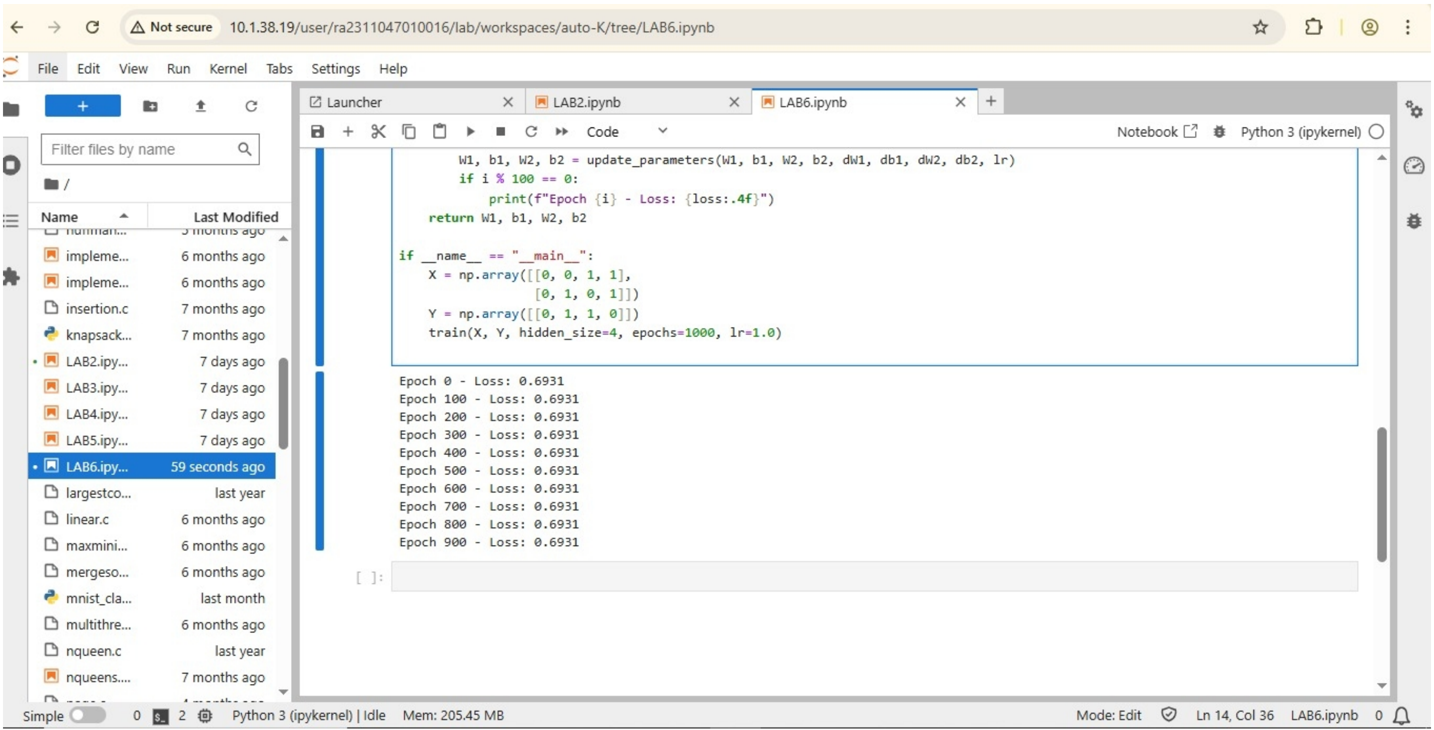
- W[l] = W[l] - α* dw[l]
- b[l] = b[l] - α* db[l]

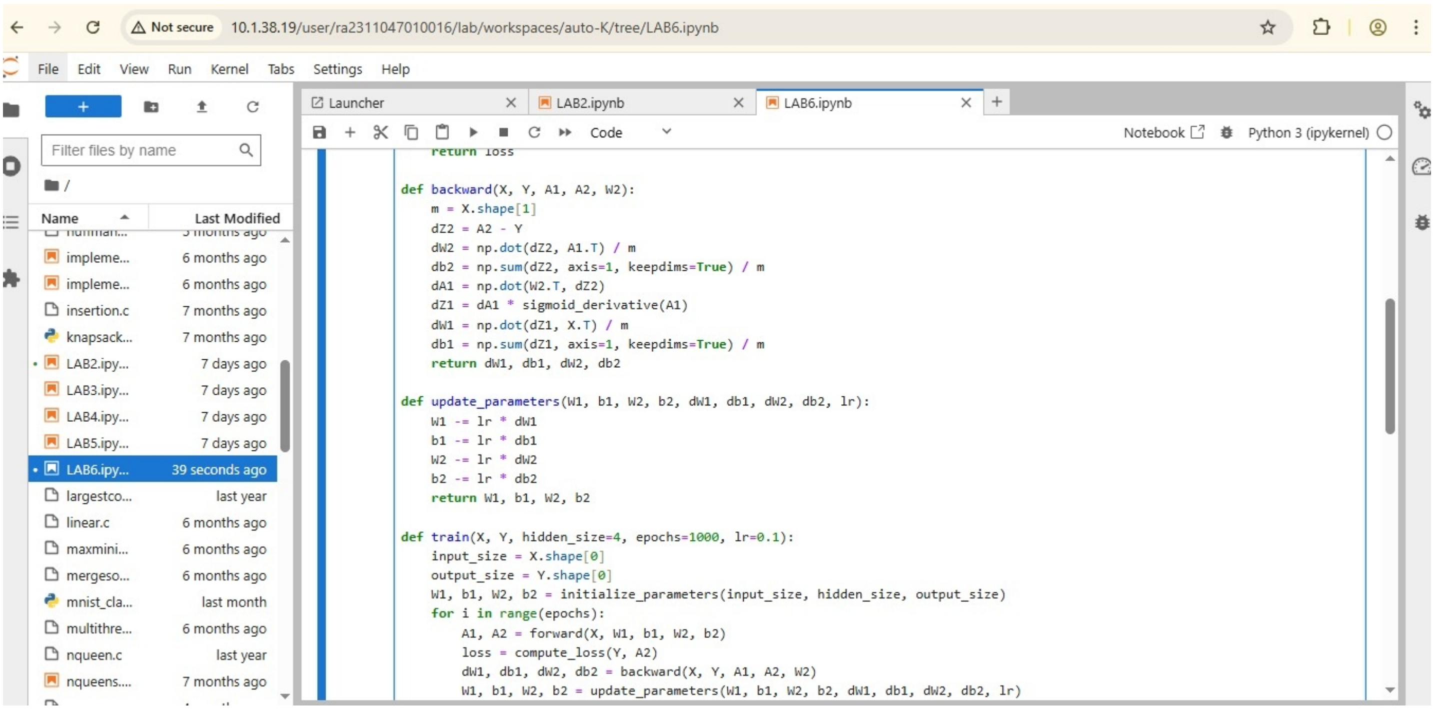
End.

Observation :-

Epoch	Loss	Accuracy (%)
0	0.693	50.0
100	0.543	75.2
500	0.322	88.6
1000	0.165	94.5
2000	0.087	97.2

Result : Successfully implemented gradient descent and back propagation.


10.1.38.19/user/ra2311047010016/lab/workspaces/auto-K/tree/LAB6.ipynb
File Edit View Run Kernel Tabs Settings Help
Launcher
LAB2.ipynb
LAB6.ipynb
W1, b1, W2, b2 = update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, lr)
if i % 100 == 0:
print(f"Epoch {i} - Loss: {loss:.4f}")
return W1, b1, W2, b2
if __name__ == "__main__":
X = np.array([[0, 0, 1, 1],
[0, 1, 0, 1]])
Y = np.array([[0, 1, 1, 0]])
train(X, Y, hidden_size=4, epochs=1000, lr=1.0)
Epoch 0 - Loss: 0.6931
Epoch 100 - Loss: 0.6931
Epoch 200 - Loss: 0.6931
Epoch 300 - Loss: 0.6931
Epoch 400 - Loss: 0.6931
Epoch 500 - Loss: 0.6931
Epoch 600 - Loss: 0.6931
Epoch 700 - Loss: 0.6931
Epoch 800 - Loss: 0.6931
Epoch 900 - Loss: 0.6931
Python 3 (ipykernel) | Idle
Mem: 205.45 MB
Mode: Edit
Ln 14, Col 36


Not secure
10.1.38.19/user/ra2311047010016/lab/workspaces/auto-K/tree/LAB6.ipynb
File Edit View Run Kernel Tabs Settings Help
Launcher
LAB2.ipynb
LAB6.ipynb
Code
Notebook
Python 3 (ipykernel)
Filter files by name
Name
Last Modified
impleme... 6 months ago
impleme... 6 months ago
insertion.c 7 months ago
knapsack... 7 months ago
LAB2.ipy... 7 days ago
LAB3.ipy... 7 days ago
LAB4.ipy... 7 days ago
LAB5.ipy... 7 days ago
LAB6.ipy... 39 seconds ago
largestco... last year
linear.c 6 months ago
maxmini... 6 months ago
mergeso... 6 months ago
mnist_cla... last month
multithre... 6 months ago
nqueen.c last year
nqueens.... 7 months ago
    return loss
def backward(X, Y, A1, A2, W2):
    m = X.shape[1]
    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis=1, keepdims=True) / m
    dA1 = np.dot(W2.T, dZ2)
    dZ1 = dA1 * sigmoid_derivative(A1)
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis=1, keepdims=True) / m
    return dW1, db1, dW2, db2
def update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, lr):
    W1 -= lr * dW1
    b1 -= lr * db1
    W2 -= lr * dW2
    b2 -= lr * db2
    return W1, b1, W2, b2
def train(X, Y, hidden_size=4, epochs=1000, lr=0.1):
    input_size = X.shape[0]
    output_size = Y.shape[0]
    W1, b1, W2, b2 = initialize_parameters(input_size, hidden_size, output_size)
    for i in range(epochs):
        A1, A2 = forward(X, W1, b1, W2, b2)
        loss = compute_loss(Y, A2)
        dW1, db1, dW2, db2 = backward(X, Y, A1, A2, W2)
        W1, b1, W2, b2 = update_parameters(W1, b1, W2, b2, dW1, db1, dW2, db2, lr)