9-9-25

LAB-6 Implement Gradient Descent & Backpropogation in Deep Neural Network

Aim: To implement gradient descent and backpropogation algorithm in deep neural network and study their role in training.

Objective:

1. To understand the working of gradient Optimization.
2. To implement backpropogation for updating neural network weights.
3) To observe the effect of iterations on loss reduction.

Pseudocode:

```
1. Initialize weights and bias randomly.
2. For each epoch:
      a) Forward pass
            - Compute weighted sum.
                  ($z = w \cdot x + b$)
            - Apply activation function
                  ($A = f(z)$)
      b) Compute loss
            L = diffrence b/w predicted and actual.

      c) Backward pass:
            - Compute gradients of loss w.r.t weights
              & biases.
            - update weights: $w = w - \eta * dL/db$
```

3) Repeat until Converges.

Formula used

$Z = W \cdot x + b$

$L = 1/n \sum (y - \hat{y})^2$

$W = W - \eta \frac{dL}{dW}$

Observation:

1. Initially, the model started with random weights leading to high loss and low accuracy.

2. with each epoch gradient descent gradually reduced the loss, showing effects of iterative weight update.

3) Backpropogation effectively adjusted weights layer by layer, improving model accuracy.

4. The choice of learning rate and number of epochs strongly influenced convergence speed and final accuracy.

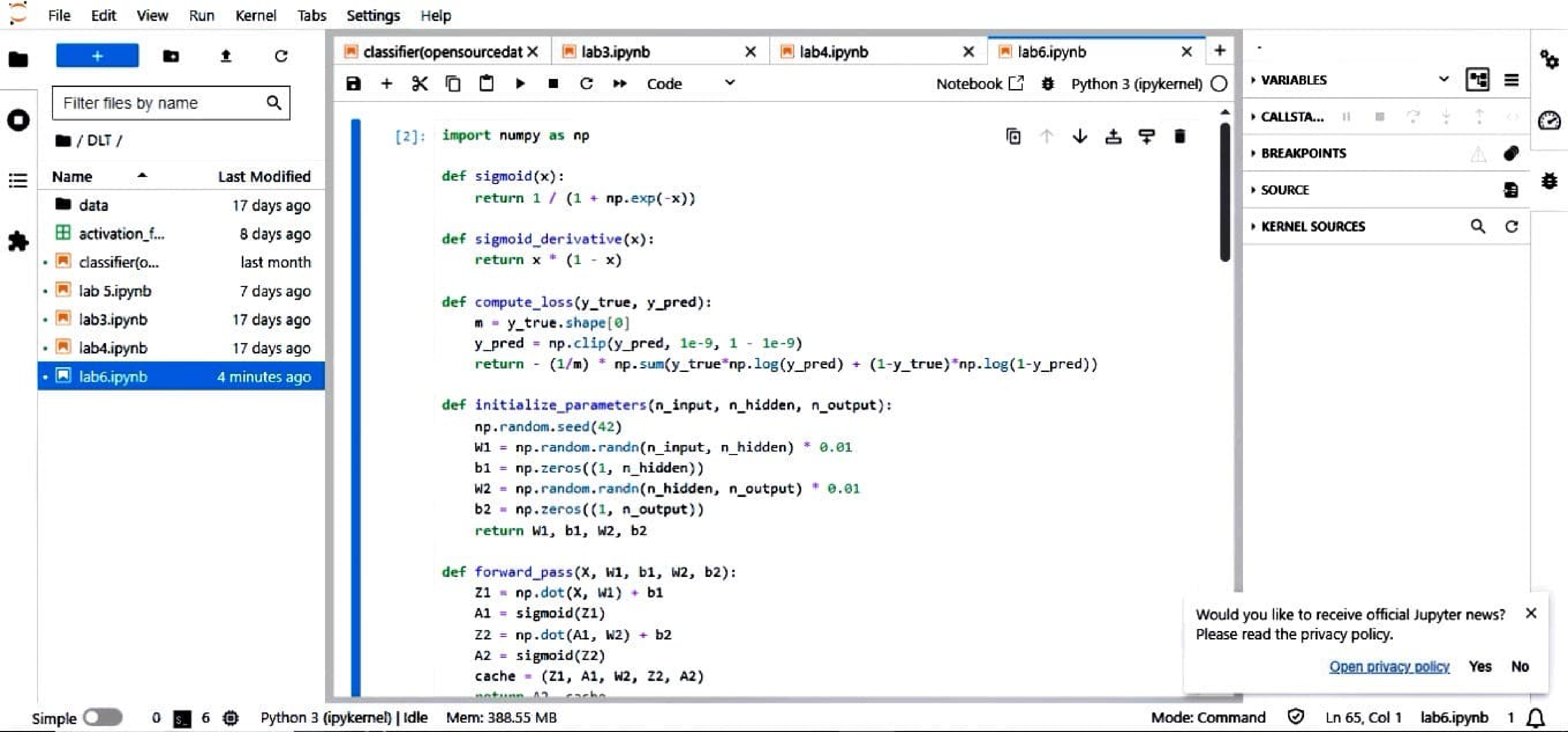
Sample:

Epochs	Training loss	Accuracy	Remark
1	0.95	65.0	High error
5	0.48	88.3	loss decreasing
10	0.25	90.1	faster convergence
20	0.12	95.5	model stabilized.

Page
Date

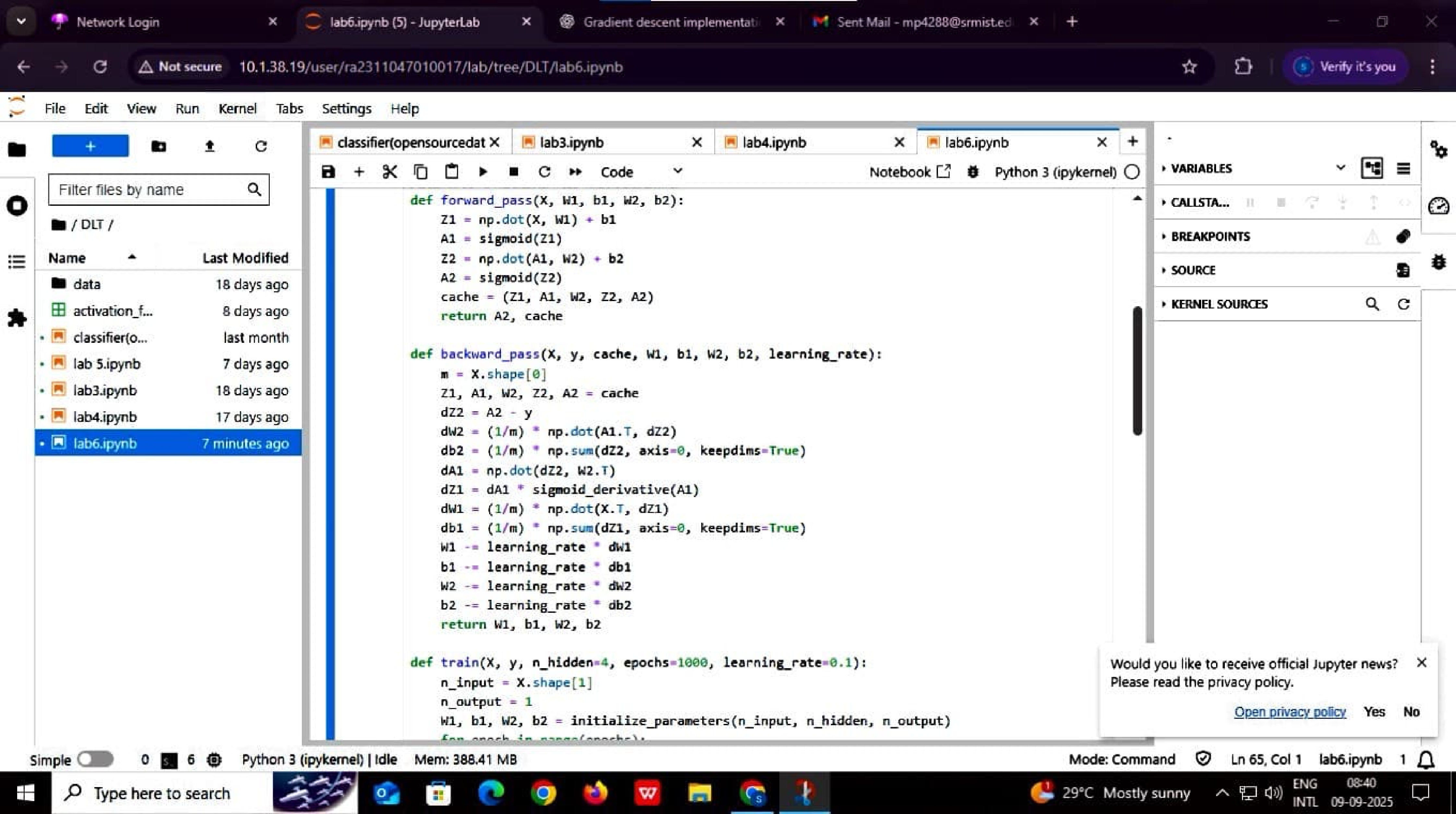
Result :

Implemented Gradient descent & Backpropogation in DNN.



[2]:

```
import numpy as np

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

def sigmoid_derivative(x):
    return x * (1 - x)

def compute_loss(y_true, y_pred):
    m = y_true.shape[0]
    y_pred = np.clip(y_pred, 1e-9, 1 - 1e-9)
    return - (1/m) * np.sum(y_true*np.log(y_pred) + (1-y_true)*np.log(1-y_pred))

def initialize_parameters(n_input, n_hidden, n_output):
    np.random.seed(42)
    W1 = np.random.randn(n_input, n_hidden) * 0.01
    b1 = np.zeros((1, n_hidden))
    W2 = np.random.randn(n_hidden, n_output) * 0.01
    b2 = np.zeros((1, n_output))
    return W1, b1, W2, b2

def forward_pass(X, W1, b1, W2, b2):
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)
    cache = (Z1, A1, W2, Z2, A2)
    return A2, cache
```

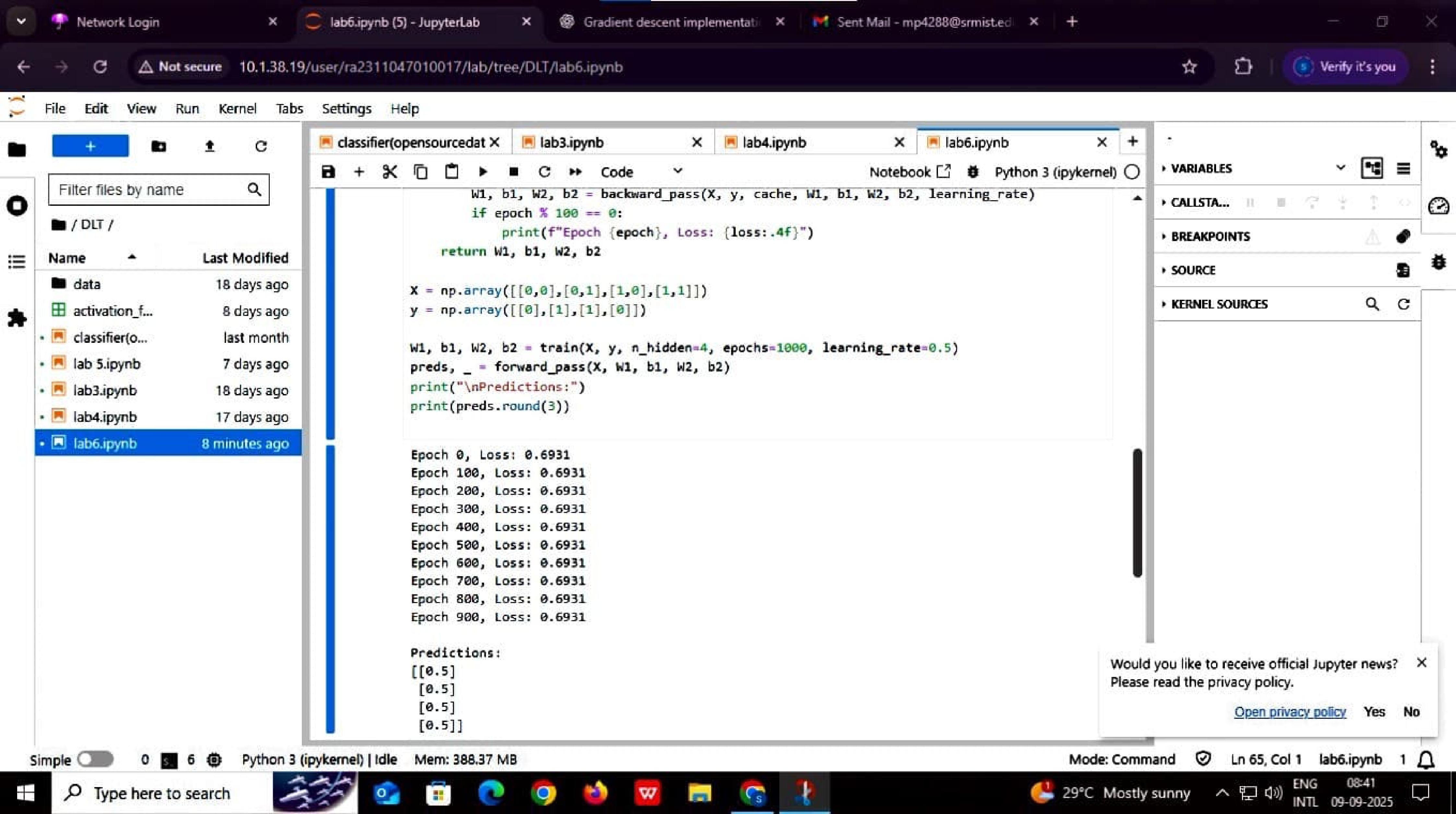


Name	Last Modified
data	18 days ago
activation_f...	8 days ago
classifier(o...	last month
lab 5.ipynb	7 days ago
lab3.ipynb	18 days ago
lab4.ipynb	17 days ago
lab6.ipynb	7 minutes ago

```
def forward_pass(X, W1, b1, W2, b2):
    Z1 = np.dot(X, W1) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = sigmoid(Z2)
    cache = (Z1, A1, W2, Z2, A2)
    return A2, cache

def backward_pass(X, y, cache, W1, b1, W2, b2, learning_rate):
    m = X.shape[0]
    Z1, A1, W2, Z2, A2 = cache
    dZ2 = A2 - y
    dW2 = (1/m) * np.dot(A1.T, dZ2)
    db2 = (1/m) * np.sum(dZ2, axis=0, keepdims=True)
    dA1 = np.dot(dZ2, W2.T)
    dZ1 = dA1 * sigmoid_derivative(A1)
    dW1 = (1/m) * np.dot(X.T, dZ1)
    db1 = (1/m) * np.sum(dZ1, axis=0, keepdims=True)
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2
    return W1, b1, W2, b2

def train(X, y, n_hidden=4, epochs=1000, learning_rate=0.1):
    n_input = X.shape[1]
    n_output = 1
    W1, b1, W2, b2 = initialize_parameters(n_input, n_hidden, n_output)
    for epoch in range(epochs):
```

Would you like to receive official Jupyter news? Please read the privacy policy.

Open privacy policy Yes No



```
        W1, b1, W2, b2 = backward_pass(X, y, cache, W1, b1, W2, b2, learning_rate)
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Loss: {loss:.4f}")
    return W1, b1, W2, b2

X = np.array([[0,0],[0,1],[1,0],[1,1]])
y = np.array([[0],[1],[1],[0]])

W1, b1, W2, b2 = train(X, y, n_hidden=4, epochs=1000, learning_rate=0.5)
preds, _ = forward_pass(X, W1, b1, W2, b2)
print("\nPredictions:")
print(preds.round(3))
```

```
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

Predictions:
[[0.5]
 [0.5]
 [0.5]
 [0.5]]
```