Lab 6: Implement Gradient Descent & Backpropagation in Deep Neural Network

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

Objective :

1) To understand the working of gradient optimization
2) To implement backpropagation for updating neural network weights.
3) To observe the effect of iterations (epochs) on loss reduction.

Pseudocode

```
1) Initialize weights and bias randomly
2) For each epoch :
       a. Forward Pass :
            - compute weighted sum
                  ( $z = w * x + b$ )
            - Apply activation function
                  ( $A = f(z)$ )
```

b. Compute Loss

$L =$ difference bw predicted & actual

c. Backward pass :

- Compute gradients of loss w.r.t weights & biases
- update weights : $w = w - \eta * dL/dw$
- update biases : $b = b - \eta * dL/db$

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 effect of iterative weight update

3) Backpropagation efficiently adjusted weights layer by layer, improving model accuracy

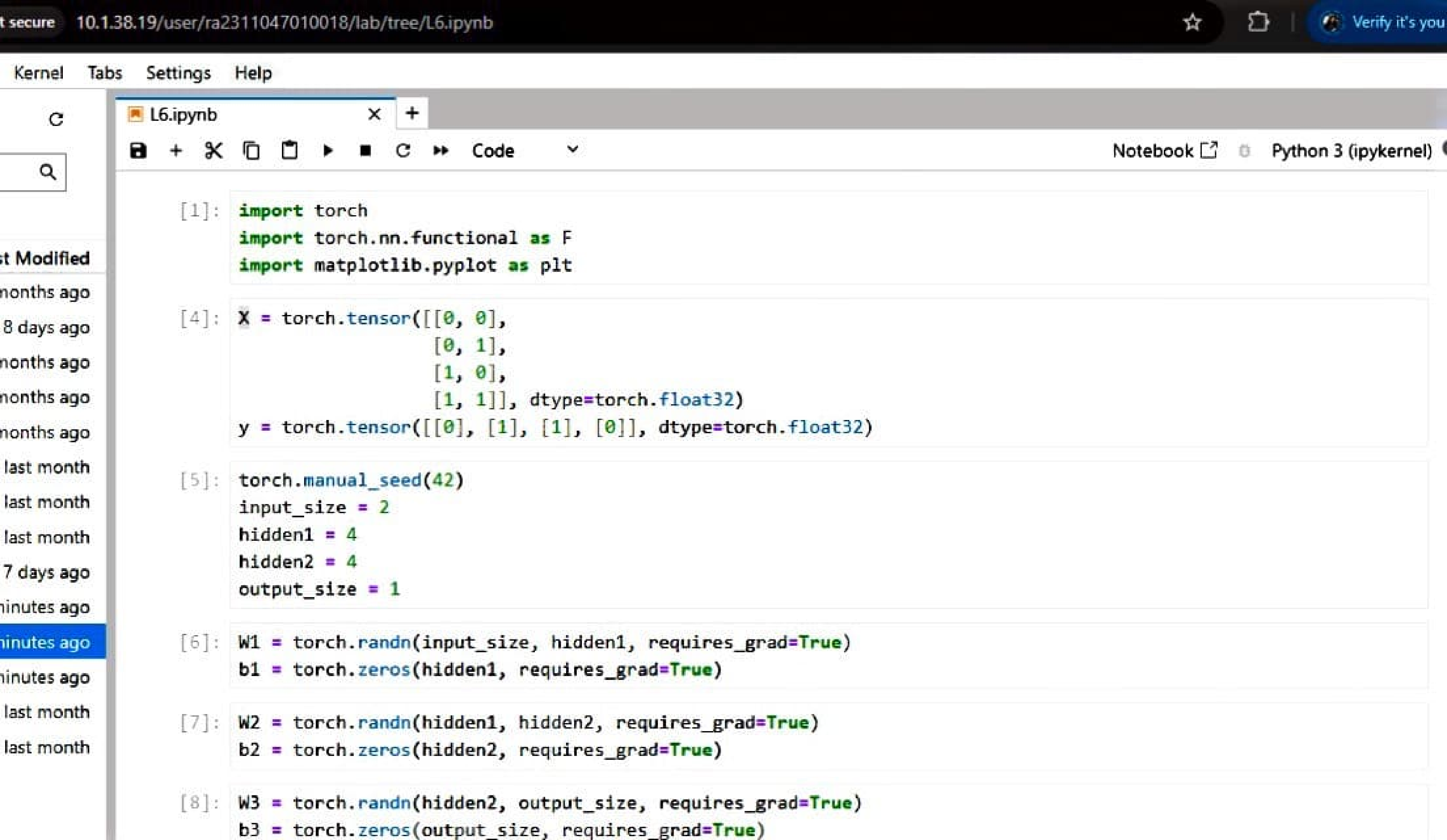
4) The choice of learning rate and nombur of epoches strongly influnced converge nce speed and final acuracy.

Sample.

Epoch	Training Loss	Accuracy	Remark
1	0.95	65.0	High error, random limit
5	0.48	82.3	Loss decreasin
10	0.25	90.1	Faster convergen
20	0.12	95.5	Model stablyzec

Result

Implemented Gradient Descent & Backpropagation in DNN



```
[1]: import torch
     import torch.nn.functional as F
     import matplotlib.pyplot as plt
```

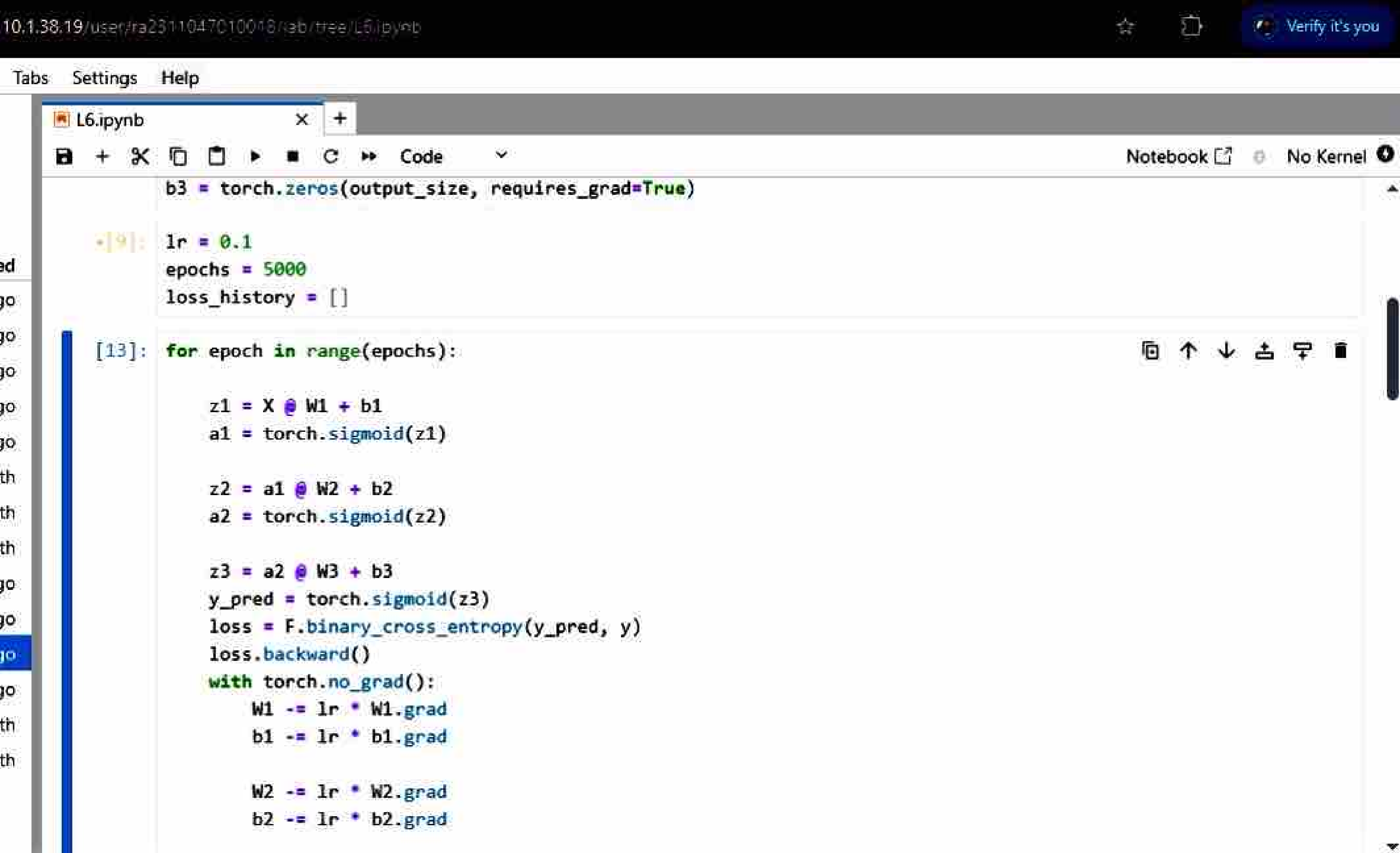
```
[4]: X = torch.tensor([[0, 0],
                       [0, 1],
                       [1, 0],
                       [1, 1]], dtype=torch.float32)
     y = torch.tensor([[0], [1], [1], [0]], dtype=torch.float32)
```

```
[5]: torch.manual_seed(42)
     input_size = 2
     hidden1 = 4
     hidden2 = 4
     output_size = 1
```

```
[6]: W1 = torch.randn(input_size, hidden1, requires_grad=True)
     b1 = torch.zeros(hidden1, requires_grad=True)
```

```
[7]: W2 = torch.randn(hidden1, hidden2, requires_grad=True)
     b2 = torch.zeros(hidden2, requires_grad=True)
```

```
[8]: W3 = torch.randn(hidden2, output_size, requires_grad=True)
     b3 = torch.zeros(output_size, requires_grad=True)
```



```
b3 = torch.zeros(output_size, requires_grad=True)
```

```
lr = 0.1
epochs = 5000
loss_history = []
```

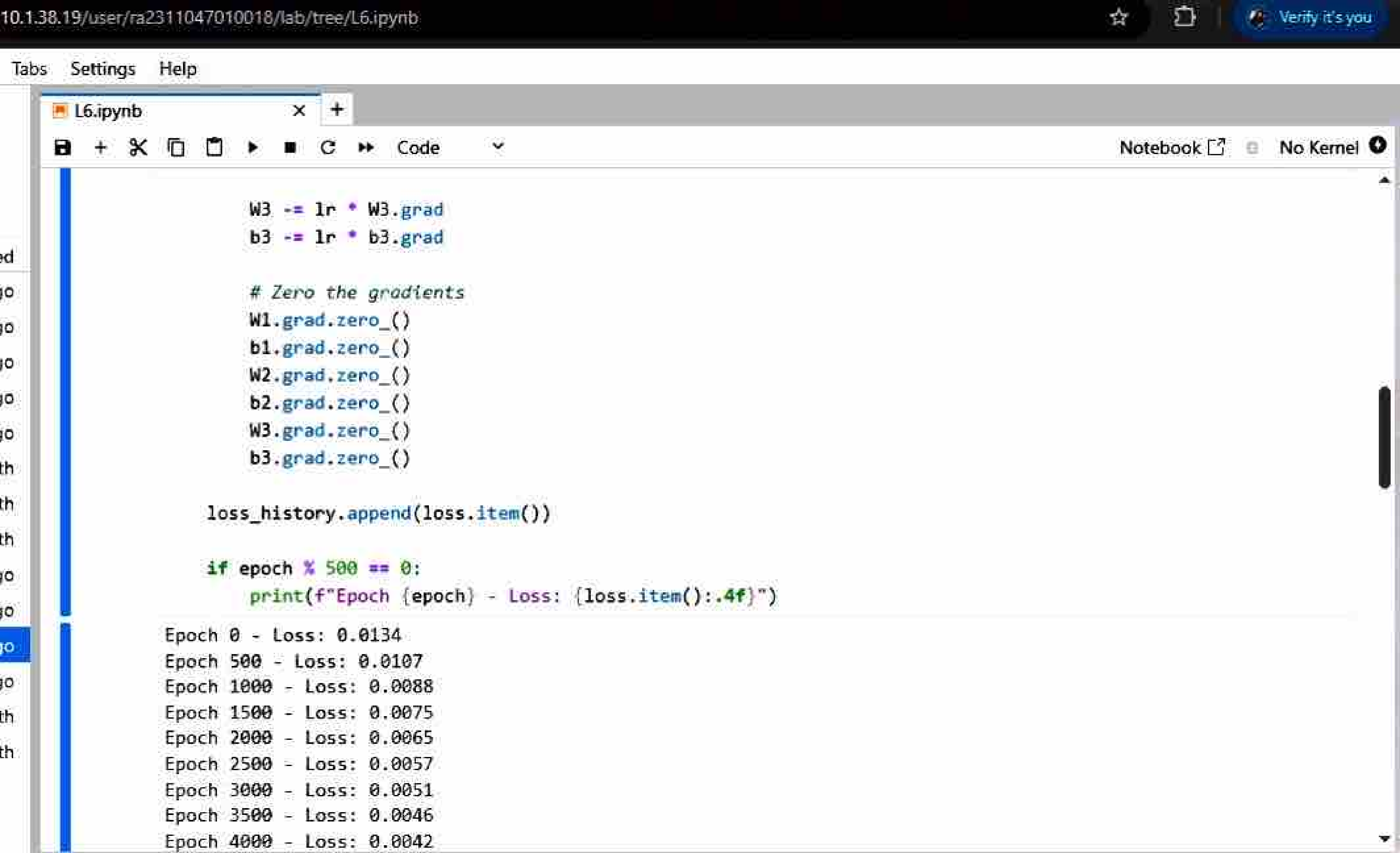
```
for epoch in range(epochs):

    z1 = X @ W1 + b1
    a1 = torch.sigmoid(z1)

    z2 = a1 @ W2 + b2
    a2 = torch.sigmoid(z2)

    z3 = a2 @ W3 + b3
    y_pred = torch.sigmoid(z3)
    loss = F.binary_cross_entropy(y_pred, y)
    loss.backward()
    with torch.no_grad():
        W1 -= lr * W1.grad
        b1 -= lr * b1.grad

        W2 -= lr * W2.grad
        b2 -= lr * b2.grad
```



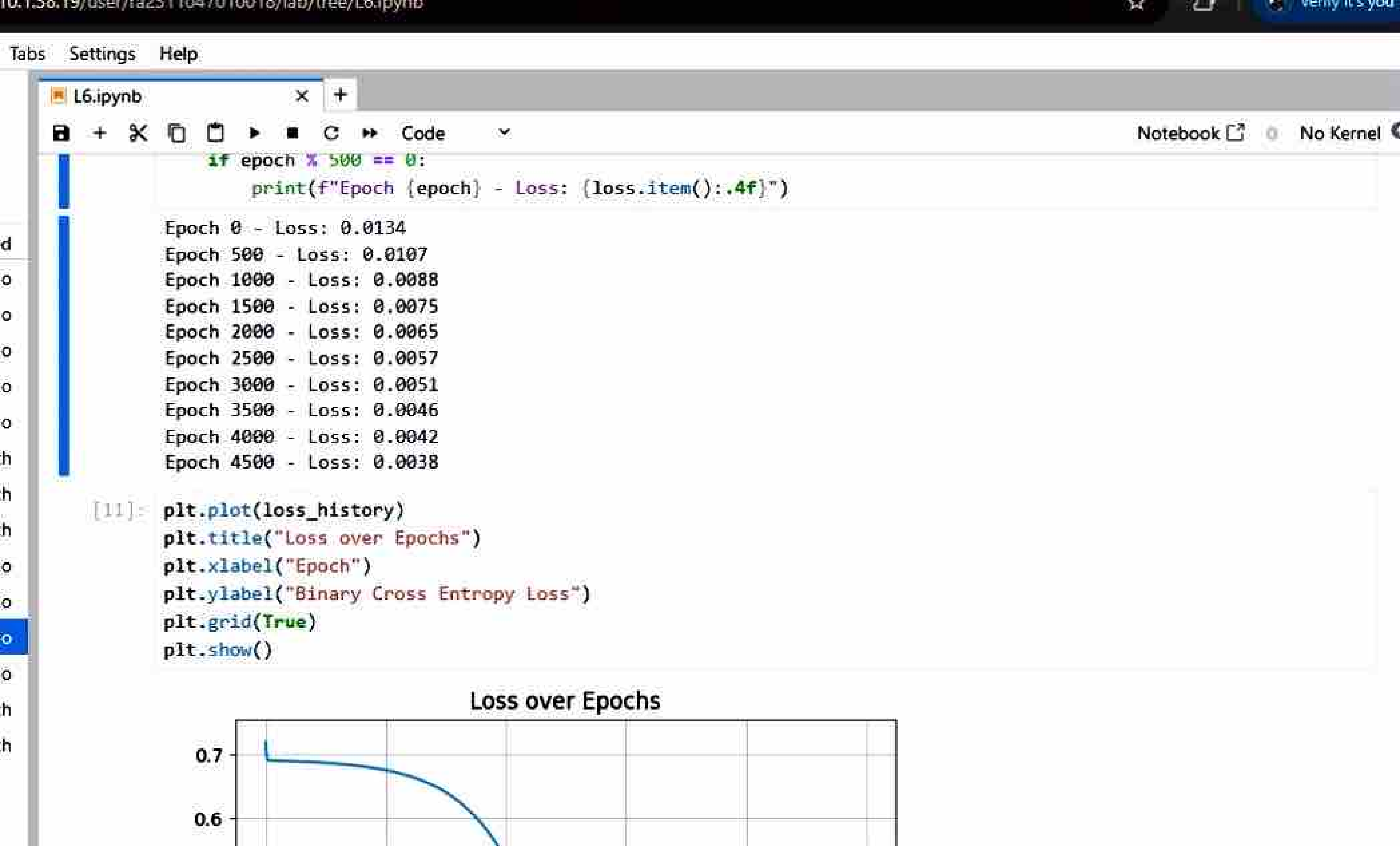
```
        W3 -= lr * W3.grad
        b3 -= lr * b3.grad

        # Zero the gradients
        W1.grad.zero_()
        b1.grad.zero_()
        W2.grad.zero_()
        b2.grad.zero_()
        W3.grad.zero_()
        b3.grad.zero_()

    loss_history.append(loss.item())

    if epoch % 500 == 0:
        print(f"Epoch {epoch} - Loss: {loss.item():.4f}")
```

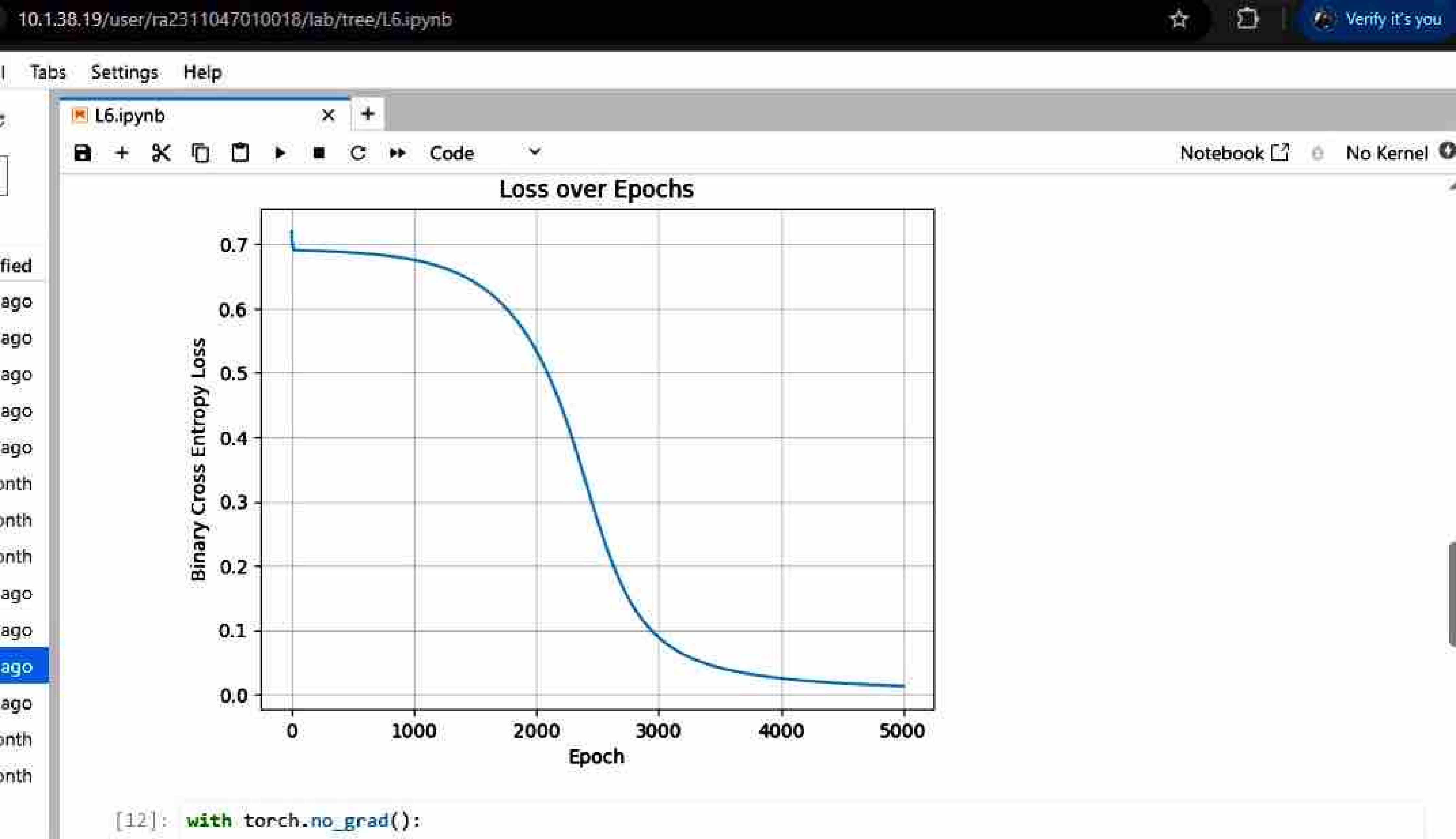
```
Epoch 0 - Loss: 0.0134
Epoch 500 - Loss: 0.0107
Epoch 1000 - Loss: 0.0088
Epoch 1500 - Loss: 0.0075
Epoch 2000 - Loss: 0.0065
Epoch 2500 - Loss: 0.0057
Epoch 3000 - Loss: 0.0051
Epoch 3500 - Loss: 0.0046
Epoch 4000 - Loss: 0.0042
```



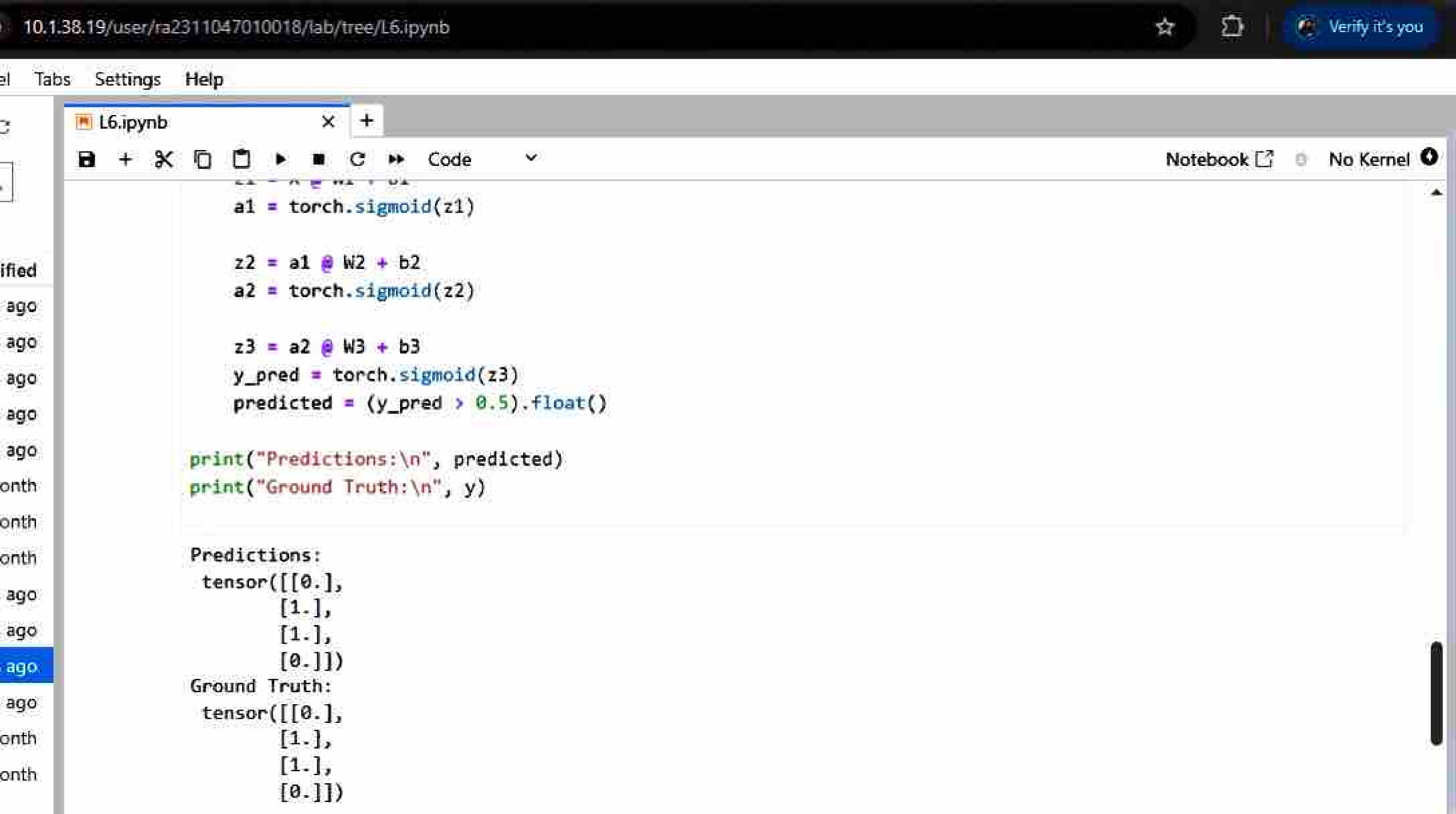
```
        if epoch % 500 == 0:
            print(f"Epoch {epoch} - Loss: {loss.item():.4f}")
```

```
Epoch 0 - Loss: 0.0134
Epoch 500 - Loss: 0.0107
Epoch 1000 - Loss: 0.0088
Epoch 1500 - Loss: 0.0075
Epoch 2000 - Loss: 0.0065
Epoch 2500 - Loss: 0.0057
Epoch 3000 - Loss: 0.0051
Epoch 3500 - Loss: 0.0046
Epoch 4000 - Loss: 0.0042
Epoch 4500 - Loss: 0.0038
```

```
plt.plot(loss_history)
plt.title("Loss over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Binary Cross Entropy Loss")
plt.grid(True)
plt.show()
```



```
[12]: with torch.no_grad():
```



```
    z1 = x @ W1 + b1
    a1 = torch.sigmoid(z1)

    z2 = a1 @ W2 + b2
    a2 = torch.sigmoid(z2)

    z3 = a2 @ W3 + b3
    y_pred = torch.sigmoid(z3)
    predicted = (y_pred > 0.5).float()

print("Predictions:\n", predicted)
print("Ground Truth:\n", y)
```

```
Predictions:
 tensor([[0.],
        [1.],
        [1.],
        [0.]])
Ground Truth:
 tensor([[0.],
        [1.],
        [1.],
        [0.]])
```