RA2311047010044

38

INDEX

NAME K.D.D.SaiAbhiram SUBJECT Deep learning lab obcb

STD. _____ DIV. _____ ROLL NO. _____ SCHOOL _____

SR. NO.	PAGE NO.	TITLE	DATE	TEACHER'S SIGN / REMARKS
1		Analyze and exploring the deep learning platforms	24-07-25	
2		Implement a classifier using open-source data set	7/8/25	
3.		Study of the classifiers with respect to statistical parameters	7/8/25	
4.		Build a simple feed forward neural network to recognize hand written characters	14/8/25	
5.		Study of activation function and its role	28-8-25	
6.		Implement gradient descent & back propagation in deep neural network.	19-9-25	

6. Implement gradient descent & back propagation in deep neural network

Aim :-

To implement gradient descent and back propagation algorithm in a simple deep neural network and study their role in training.

Objectives

1. to understand the working of gradient descent as an optimization method
2. to implement back propagation for updating weights in neural network
3. to train single neural network for a classification task using this algorithm

Pseudo Code :-

1. Start
2. Initialize dataset (x, y) for training
3. Initialize neural network parameter
   * Input layer, hidden layer, output layer
   * Random weights & biases
   * learning rate
4. forward propagation

$z_1 = w_1 * x + b_1$

$A_1 =$ activation

$A_2 =$ activation z_2

$z_2 = w_2 * A_1 + b_2$

(5) Compute loss using mean square error

(6) Back propagation

* Compute error at output
* Calculate gradients for w & b

(7) update weight and bias

$$w = w - \eta * dw$$

$$b = b - \eta * db$$

(8) Repeat the epoches

(9) observe loss and accuracy improvement

Observations

→ loss decreases as the number of iteration epoch increases

→ weights and biases adjust to minize the error

→ Learning rate (η) quantity influences convergence speed.

1, 9, 9, 15

Results

Successfully implemented gradient descent and back propagation in deep learning.

Outputs

Epoch 0 ; loss :- 0.6932 ; Accuracy : 0.3650

Epoch 200 ; loss : 0.6921 ; Accuracy : 0.5200

Epoch 300 ; loss : 0.6910 ; 0.5200

Epoch 400 ; 0.6849 ; 0.5200

Epoch 500 ; 0.6631 ; 0.5200

0.6040 ; 0.5300

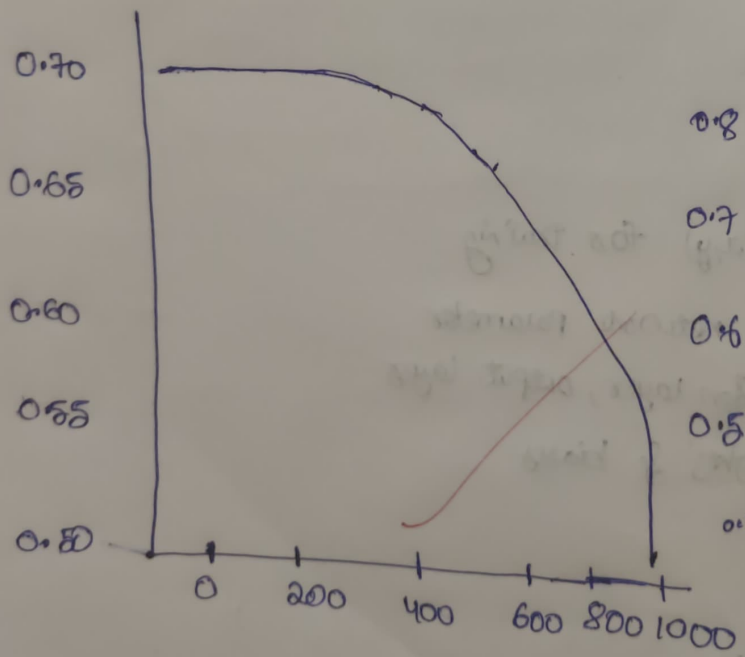
Epoch 600 ; 0.6200

Epoch 700 ; 0.6350

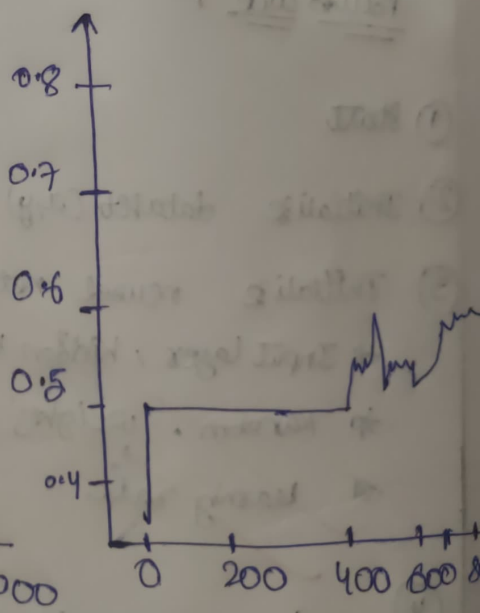
Epoch 800 ; 0.5729 ; 0.6900

Epoch 900 ; 0.5324 ; 0.8250

Training loss



Training accuracy



```python
import numpy as np
import matplotlib.pyplot as plt

# --- Create toy dataset (binary classification) ---
np.random.seed(42)
X = np.random.randn(200, 2)   # 200 samples, 2 features
y = (X[:, 0] * X[:, 1] > 0).astype(int).reshape(-1, 1)

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

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

def relu_deriv(z):
    return (z > 0).astype(float)

# --- Initialize weights ---
hidden_units = 4
W1 = np.random.randn(2, hidden_units) * 0.01
b1 = np.zeros((1, hidden_units))
W2 = np.random.randn(hidden_units, 1) * 0.01
b2 = np.zeros((1, 1))
```

```python
# --- Initialize weights ---
hidden_units = 4
W1 = np.random.randn(2, hidden_units) * 0.01
b1 = np.zeros((1, hidden_units))
W2 = np.random.randn(hidden_units, 1) * 0.01
b2 = np.zeros((1, 1))

# --- Training loop ---
lr = 0.1
epochs = 1000
losses = []       # store losses
accuracies = []   # store accuracies
epoch_list = []

for epoch in range(epochs):
    # Forward pass
    Z1 = X.dot(W1) + b1
    A1 = relu(Z1)
    Z2 = A1.dot(W2) + b2
    A2 = sigmoid(Z2)

    # Loss (binary cross entropy)
    m = X.shape[0]
    loss = -np.mean(y * np.log(A2 + 1e-8) + (1 - y) * np.log(1 - A2 + 1e-8))
    losses.append(loss)

    # Accuracy
    predictions = (A2 > 0.5).astype(int)
    accuracy = np.mean(predictions == y)
    accuracies.append(accuracy)

    epoch_list.append(epoch)

    # Backpropagation
    dZ2 = A2 - y
    dW2 = (A1.T.dot(dZ2)) / m
    db2 = np.mean(dZ2, axis=0, keepdims=True)

    dA1 = dZ2.dot(W2.T)
    dZ1 = dA1 * relu_deriv(Z1)
    dW1 = (X.T.dot(dZ1)) / m
    db1 = np.mean(dZ1, axis=0, keepdims=True)
```

```
    # Gradient descent update
    W1 -= lr * dW1
    b1 -= lr * db1
    W2 -= lr * dW2
    b2 -= lr * db2

    # Print every 100 epochs
    if epoch % 100 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.4f}, Accuracy: {accuracy:.4f}")

# --- Plot Loss & Accuracy ---
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.plot(epoch_list, losses, label="Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training Loss")
plt.legend()

plt.subplot(1,2,2)
plt.plot(epoch_list, accuracies, label="Accuracy", color="orange")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy")
plt.legend()

plt.tight_layout()
plt.show()
```

```
Epoch 0, Loss: 0.6932, Accuracy: 0.3650
Epoch 100, Loss: 0.6923, Accuracy: 0.5200
Epoch 200, Loss: 0.6921, Accuracy: 0.5200
Epoch 300, Loss: 0.6910, Accuracy: 0.5200
Epoch 400, Loss: 0.6849, Accuracy: 0.5200
Epoch 500, Loss: 0.6631, Accuracy: 0.5300
Epoch 600, Loss: 0.6318, Accuracy: 0.6200
Epoch 700, Loss: 0.6040, Accuracy: 0.6350
Epoch 800, Loss: 0.5729, Accuracy: 0.6700
Epoch 900, Loss: 0.5324, Accuracy: 0.8250
```