KESHAV MEMORIAL INSTITUTE OF TECHNOLOGY
AN AUTONOMOUS INSTITUTION - ACCREDITED BY NAAC WITH 'A' GRADE
Narayanaguda, Hyderabad.

BUILD ANN FROM SCRATCH

Exercise-2

By

Asha

1.Data Loading and Preprocessing:

We load the Boston Housing dataset, scale the features, and split the data into training and testing sets.

2.Forward Pass:

Compute the linear combination of inputs and weights for the hidden layer (Z1).

Apply the sigmoid activation function to get the activations of the hidden layer (A1).

Compute the linear combination for the output layer (Z2).

3.Loss Calculation:

Compute the Mean Squared Error (MSE) loss between the predicted and actual target values.

4.Backpropagation:

Compute the gradients of the loss with respect to the weights and biases of the output layer.

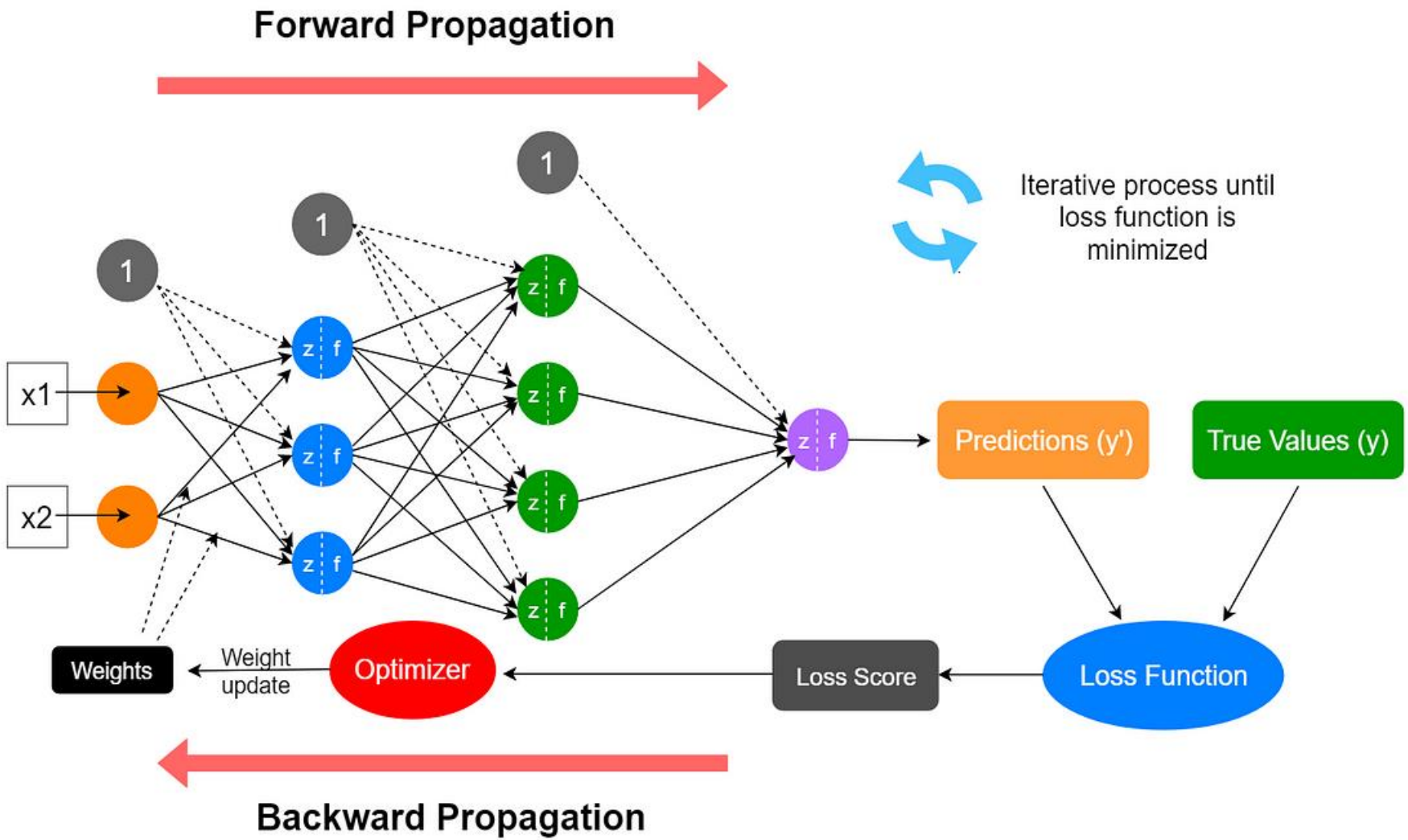
Compute the gradients for the hidden layer weights and biases using the chain rule and sigmoid derivative.

Update the weights and biases using gradient descent.

5.Training and Testing:

Train the ANN on the training data and print the training loss every 100 epochs.

Evaluate the model on the test data and print the test loss.


Forward Propagation
1
1
1
x1
x2
z f
z f
z f
z f
z f
z f
z f
Iterative process until loss function is minimized
Predictions (y')
True Values (y)
Loss Function
Loss Score
Optimizer
Weight update
Weights
Backward Propagation

You are tasked with creating a simple artificial neural network (ANN) from scratch to find price of the house using boston data using the Scikit-learn boston dataset. Write a Python script that includes the following steps:

A. Data Loading and preprocessing

1.Import Necessary libraries

2.# Load the Boston Housing dataset and Split the dependent and independent variables(X,y)

3.# Standardize the features

4.# Split the dataset into training and testing sets

B. Neural Network Implementation

1.# Define the architecture

2.# Initialize weights and biases

3.# Define Activation function (ReLU)

4.# Define Derivative of ReLU

5.# Mean squared error loss

6.# Forward propagation

7.# Backward propagation

Compute the gradients

8.# Update parameters

C. Neural Network Training

1.# Training loop

2.# Predictions

D. Evaluate the model

1. **Loading and Preprocessing the Data**

Load the Boston housing dataset and preprocess it by splitting it into training and testing sets and scaling the features. Implement the necessary code for these steps.

Mathematical Concepts:

- **Splitting Data**: Training set (80%), Testing set (20%)
- **Standardizing Features**: $X_{\text{scaled}} = \frac{X - \mu}{\sigma}$

2. **ReLU Activation Function**

Implement the ReLU (Rectified Linear Unit) activation function and its derivative. Explain how these functions transform the input values mathematically.

Mathematical Formulas:

- ReLU Activation: $A = \max(0, Z)$
- ReLU Derivative: $\mathrm{dReLU} = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{otherwise} \end{cases}$

3. Mean Squared Error Loss Function

Define the Mean Squared Error (MSE) loss function. Calculate the loss between the true and predicted values.

Mathematical Formula:

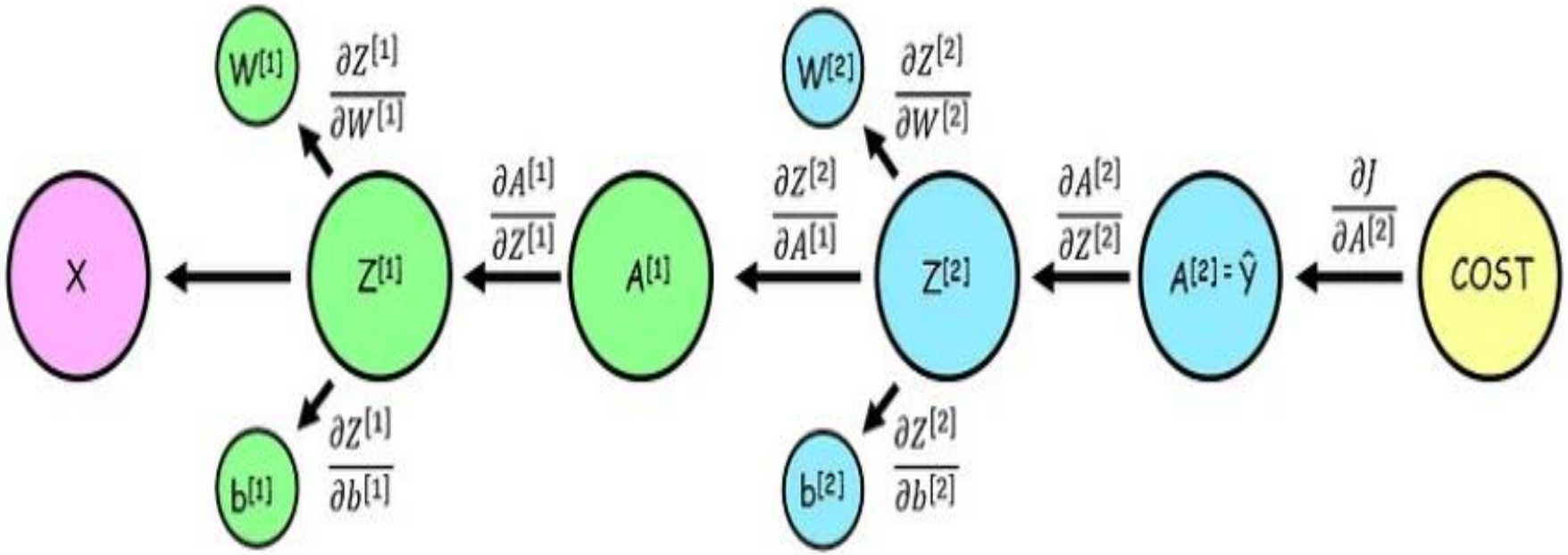
$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

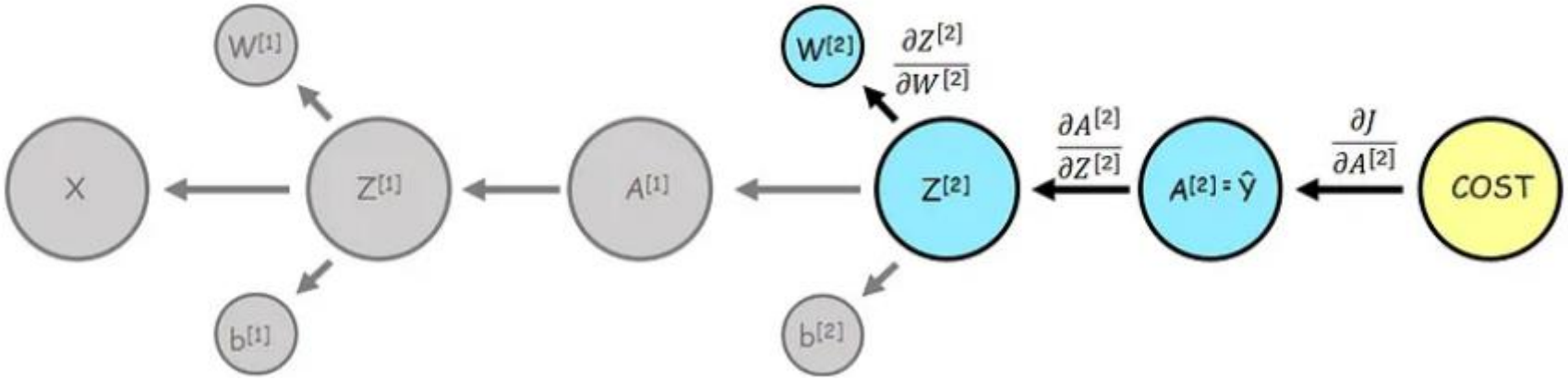
4. Forward Propagation

Implement the forward propagation step of a neural network with one hidden layer. Describe the formulas for the linear transformations and activation functions used.

Mathematical Formulas:

- First Layer: $Z_1 = XW_1 + b_1$
- ReLU Activation: $A_1 = \text{ReLU}(Z_1)$
- Second Layer: $Z_2 = A_1W_2 + b_2$


W[1]
∂Z[1] / ∂W[1]
X
Z[1]
∂A[1] / ∂Z[1]
A[1]
b[1]
∂Z[1] / ∂b[1]
W[2]
∂Z[2] / ∂W[2]
∂Z[2] / ∂A[1]
Z[2]
∂A[2] / ∂Z[2]
A[2] = ŷ
b[2]
∂Z[2] / ∂b[2]
∂J / ∂A[2]
COST



From the diagram above we can clearly see that the change in the cost J with respect to W[2] is:

$$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial W^{[2]}}$$

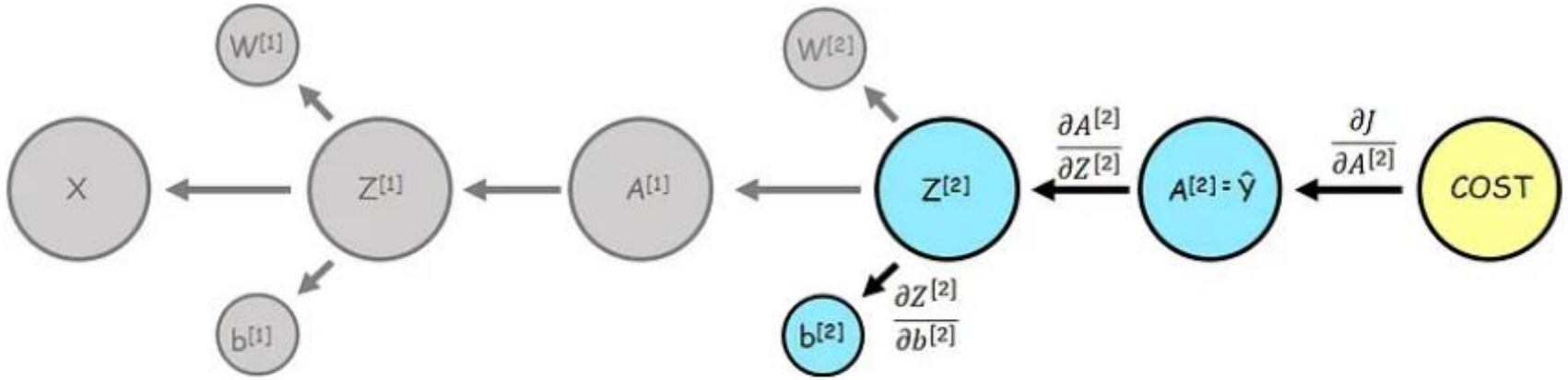


Figure 6. Gradient of the cost J with respect to the bias of layer two b[2] | Image by Author

$$\frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \frac{\partial A^{[2]}}{\partial Z^{[2]}} \frac{\partial Z^{[2]}}{\partial b^{[2]}}$$

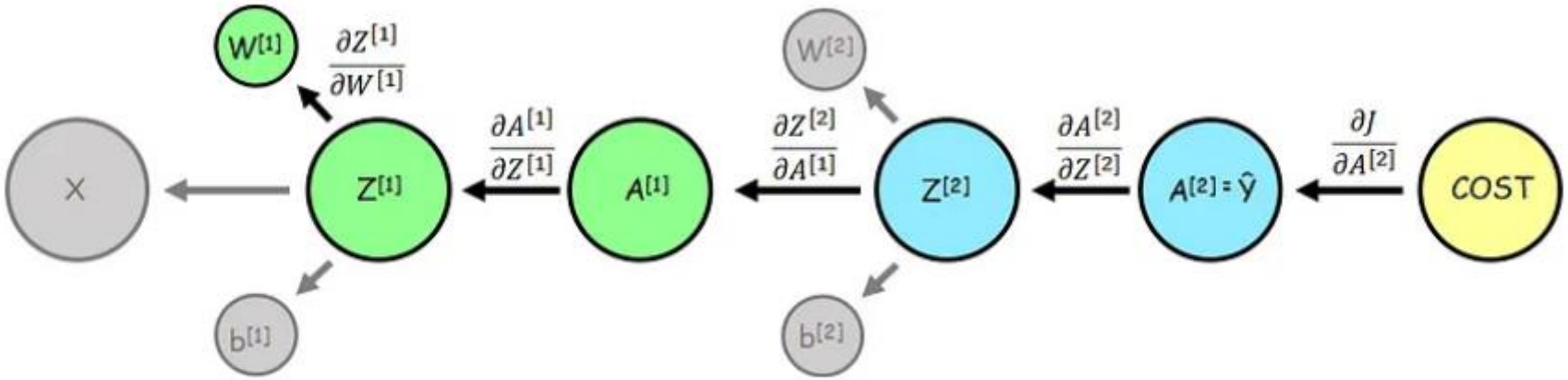


Figure 7. W[1] gradient | Image by Author

The first two parts of the gradient were previously calculated for layer 2. The partial derivative of Z[2] with respect to A[1], is W[2]:

Original function:	Partial derivative:
$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$	$\frac{\partial Z^{[2]}}{\partial A^{[1]}} = W^{[2]}$

And if we follow the b[1] gradient:

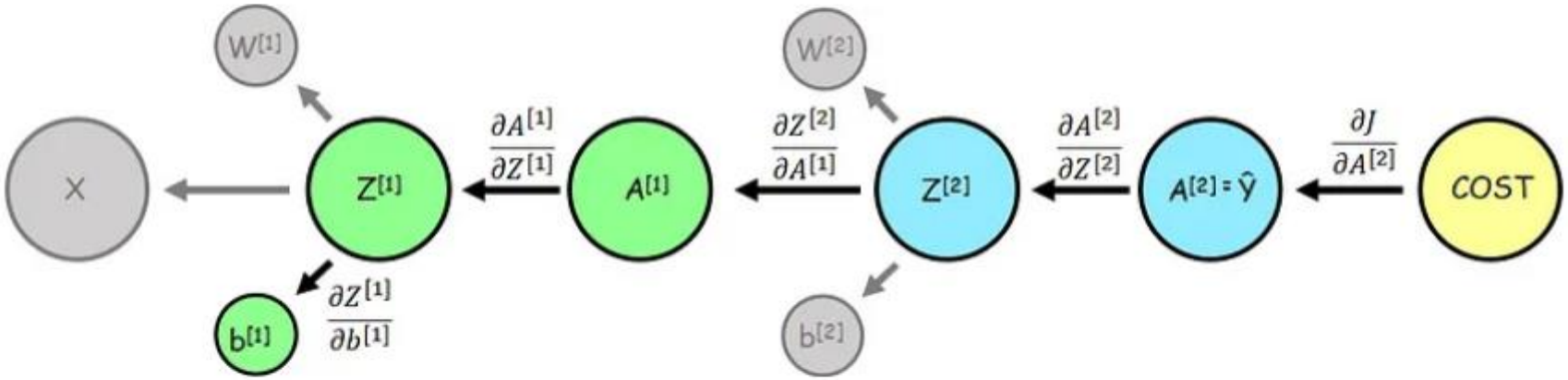


Figure 8. b[1] gradient | Image by Author

5. Backward Propagation

Implement the backward propagation step to compute gradients for updating weights and biases.

Derive the gradients with respect to each parameter.

Mathematical Formulas:

- **Gradient of Loss with Respect to Z_2:** $dZ_2 = \frac{\partial \text{Loss}}{\partial Z_2} = Z_2 - y$
- **Gradients for W_2 and b_2:**
  $dW_2 = \frac{1}{m} A_1^T dZ_2$
  $db_2 = \frac{1}{m} \sum_{i=1}^{m} dZ_2[i]$
- **Gradient for A_1:**
  $dA_1 = dZ_2 W_2^T$
- **Gradient of ReLU Activation:** $dZ_1 = dA_1 \times \text{dReLU}(Z_1)$
- **Gradients for W_1 and b_1:**
  $dW_1 = \frac{1}{m} X^T dZ_1$
  $db_1 = \frac{1}{m} \sum_{i=1}^{m} dZ_1[i]$

Kmit

Summary of Derivatives

- $\delta_2 = \frac{\partial \mathcal{L}}{\partial Z2} = \hat{Y} - Y$
- $dW2 = \frac{1}{m}\delta_2 \cdot A1^T$
- $db2 = \frac{1}{m}\sum_{i=1}^{m}\delta_2$
- $\delta_1 = \frac{\partial \mathcal{L}}{\partial Z1} = (W2^T \cdot \delta_2) \cdot \sigma'(Z1)$
- $dW1 = \frac{1}{m}\delta_1 \cdot X^T$
- $db1 = \frac{1}{m}\sum_{i=1}^{m}\delta_1$

6. **Parameter Update**

Update the weights and biases of the neural network using the gradients computed during backpropagation. Apply gradient descent to update the parameters.

Mathematical Formulas:

- **Weight Update:** $W = W - \alpha \cdot dW$
- **Bias Update:** $b = b - \alpha \cdot db$

  Where α is the learning rate.

7. Training Loop

Implement the training loop for the neural network. Iterate through the training data, perform forward and backward propagation, and update the parameters for each epoch.

Mathematical Formulas:

- Loss Calculation: $\text{Loss} = \frac{1}{m} \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$
- Parameter Update: $W = W - \alpha \cdot dW$

  $b = b - \alpha \cdot db$

8. Testing the Model

Evaluate the performance of the trained neural network on the test set. Calculate and display the test loss to assess how well the model generalizes to new data.

Mathematical Formula:

- Test Loss Calculation: $\text{Test Loss} = \frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} (y_i - \hat{y}_i)^2$