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

BUILD ANN FROM SCRATCH
Regression
Exercise-1
09-09-2024

1.Data Loading and Preprocessing:

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

2.Forwardpropogation:

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.

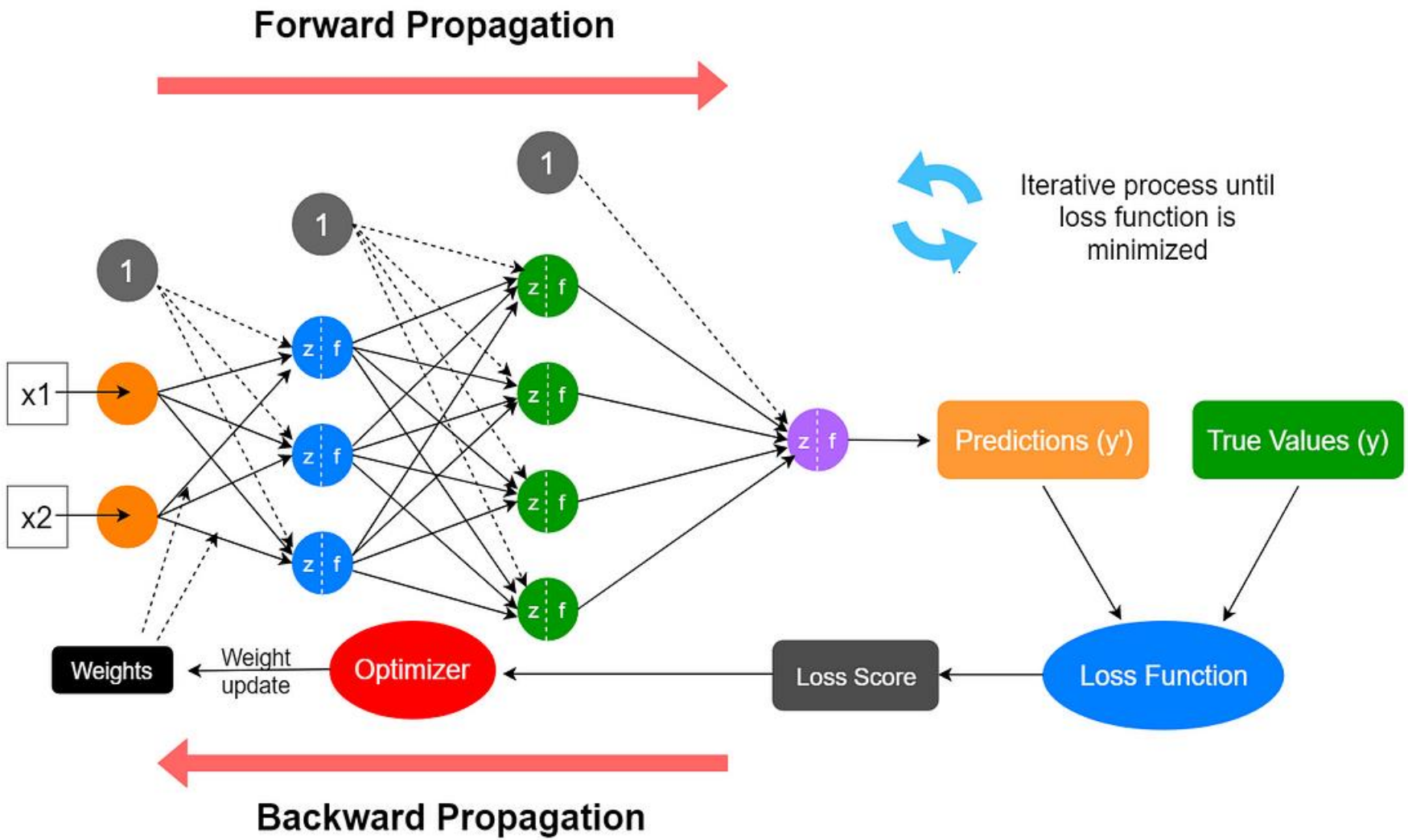
Compute the gradients for the hidden layer weights and biases using the chain rule and sigmoid derivative.

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 data into 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 # Forward propagation

6 # Mean squared error loss

7# Backward propagation

Compute the gradients

8.# Update parameters

C . 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: $\mathrm{dReLU} = \begin{cases} 1 & \text{if } Z > 0 \\ 0 & \text{otherwise} \end{cases}$

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.

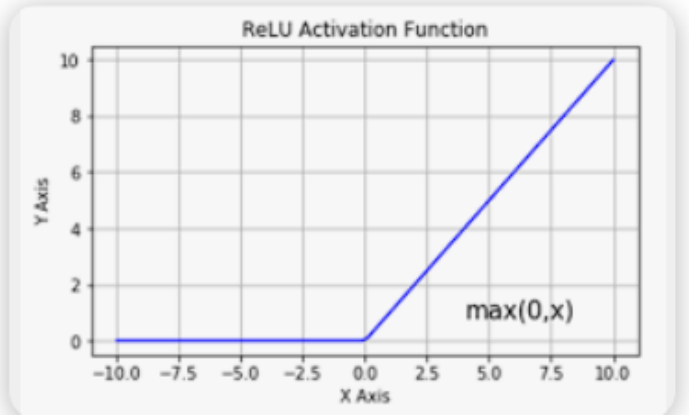
Mathematical Formulas:

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

Forward Propagation

1. Forward Step for Hidden Layer:

$$Z1 = W1 \cdot X + b1$$



$$A1 = \text{ReLU}(Z1) = \max(0, Z1)$$

2. Forward Step for Output Layer:

$$Z2 = W2 \cdot A1 + b2$$

$$\hat{Y} = Z2 \quad \text{(since it's a regression problem)}$$

Why You Might Skip the Identity Function:

If you don't use the identity function, you're essentially doing the same thing — passing the value `z` directly. So, the need for it comes down to maintaining a **clear, structured design** and **potential future modifications**. It doesn't have an immediate impact on the model's performance but helps for code readability and maintaining consistency in activation layers.

Key Points:

- **Technically**, you don't need the identity function because passing the raw value without it has the same effect.
- **Practically**, it's useful for keeping a consistent and readable model structure, especially when designing deep models or when other layers always apply some non-linear transformations.

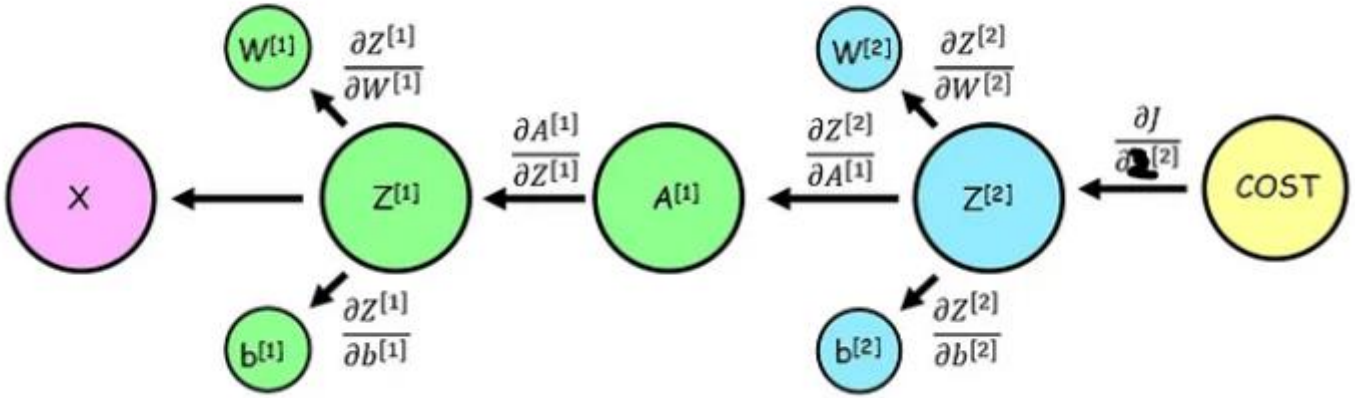
Loss Function

Mean Squared Error (MSE) loss:

$$\mathcal{L} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{Y}^{(i)} - Y^{(i)} \right)^2$$

where:

- $\hat{Y}^{(i)}$ is the predicted value for the i-th sample.
- $Y^{(i)}$ is the true value for the i-th sample.
- m is the number of samples.



Parameters:

- **X**: Input data
- **y**: True labels, shape
- **Z1**: Linear output of the hidden layer before activation
- **A1**: Activation output from the hidden layer
- **Z2**: : Linear output of the output layer before activation
- **A2**: Final output (predicted values)

Backward Propagation

1. Compute the Derivative of the Loss with Respect to $\hat{Y}$

For MSE:

$$\frac{\partial \mathcal{L}}{\partial \hat{Y}} = \frac{2}{m}(\hat{Y} - Y)$$

Let's denote this as:

$$\delta_2 = \frac{2}{m}(\hat{Y} - Y)$$

$$= \hat{Y}^{(i)} - Y^{(i)}$$

In practice, when implementing backpropagation, the factor 2/m is sometimes omitted for simplicity.

Simplified Expression

In practice, especially in the context of implementing gradient descent, the factor $\frac{2}{m}$ is sometimes omitted or factored into the learning rate. This results in the simplified gradient expression used for updating the weights:

$$\frac{\partial \mathcal{L}}{\partial \hat{Y}^{(i)}} = \hat{Y}^{(i)} - Y^{(i)}$$

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:**
  $dW_1 = \frac{1}{m} X^T dZ_1$
  $db_1 = \frac{1}{m} \sum_{i=1}^{m} dZ_1[i]$

1. **Forward Pass**:
    - Compute the values Z_1, A_1, and Z_2.
    - Calculate the MSE loss.
2. **Backward Pass**:
    - **Gradient with respect to Z_2**: $\frac{\partial \text{Loss}}{\partial Z_2} = Z_2 - Y$
    - **Gradient with respect to W_2**: $\frac{\partial \text{Loss}}{\partial W_2} = (Z_2 - Y)A_1^T$
    - **Gradient with respect to b_2**: $\frac{\partial \text{Loss}}{\partial b_2} = \sum(Z_2 - Y)$
    - **Gradient with respect to A_1**: $\frac{\partial \text{Loss}}{\partial A_1} = W_2^T(Z_2 - Y)$
    - **Gradient with respect to Z_1**: $\frac{\partial \text{Loss}}{\partial Z_1} = (W_2^T(Z_2 - Y)) \odot \text{ReLU}'(Z_1)$
    - **Gradient with respect to W_1**: $\frac{\partial \text{Loss}}{\partial W_1} = ((W_2^T(Z_2 - Y)) \odot \text{ReLU}'(Z_1))X^T$
    - **Gradient with respect to b_1**: $\frac{\partial \text{Loss}}{\partial b_1} = ((W_2^T(Z_2 - Y)) \odot \text{ReLU}'(Z_1)) \cdot \mathbf{1}$

Step 1: Compute dZ2 (Output layer error)
python
dZ2 = A2 - y
•This computes the error in the output layer (i.e., the difference between the predicted output A2 and the true label y).

Step 2: Compute gradients for W2 and b2
•**dW2**: The gradient of the cost function with respect to the weights of the output layer.
This computes how much the weights from the hidden layer to the output layer should be adjusted, based on the hidden layer activations A1 and the error dZ2.

db2: The gradient of the cost function with respect to the bias of the output layer.
In neural networks, biases are scalar values added to the neurons of a layer. During backpropagation, we calculate the gradient of the loss with respect to the biases to update them. Here, db2 is the gradient of the bias in the second layer (the output layer in this case).

This computes how much the bias term for the output layer should be adjusted based on the error dZ2.

Step 3: Compute dA1 (Error propagated to the hidden layer)
•This propagates the error back from the output layer to the hidden layer, using the weight matrix W2. This tells us how much each hidden unit contributed to the output error.

Step 4: Compute dZ1 (Hidden layer error)

•This calculates the error in the hidden layer. Since we use a ReLU activation function in the hidden layer, we apply the **derivative of the ReLU function** to the propagated error dA1. This ensures that the error is only propagated through neurons that are active (i.e., where Z1 > 0).

step 5: Compute gradients for W1 and b1

•**dW1**: The gradient of the cost function with respect to the weights of the hidden layer.
This computes how much the weights from the input layer to the hidden layer should be adjusted, based on the input X and the hidden layer error dZ1.

•**db1**: The gradient of the cost function with respect to the bias of the hidden layer.
This computes how much the bias term for the hidden layer should be adjusted, based on the error dZ1.

1. **Gradient of Loss with Respect to Z_2:**

$$dZ_2 = A_2 - y = Z_2 - y$$

Since $A_2 = Z_2$, this remains as the original difference between the output and the actual labels.

2. **Gradients for W_2 and b_2:**

$$dW_2 = \frac{1}{m} A_1^T dZ_2$$

$$db_2 = \frac{1}{m} \sum_{i=1}^{m} dZ_2[i]$$

These calculate the gradients for the weights and biases in the output layer, based on the error at the output.

3. **Gradient for A_1:**

$$dA_1 = dZ_2 W_2^T$$

This propagates the error back to the hidden layer.

4. **Gradient of ReLU Activation:**

$$dZ_1 = dA_1 \times dReLU(Z_1)$$

This applies the chain rule to propagate the error through the ReLU activation of the hidden layer.

5. **Gradients for W_1 and b_1:**

$$dW_1 = \frac{1}{m} X^T dZ_1$$

$$db_1 = \frac{1}{m} \sum_{i=1}^{m} dZ_1[i]$$

These update the weights and biases in the hidden layer based on the backpropagated error.

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.

- **Gradient Descent**: Each weight and bias is updated by subtracting the gradient (which indicates the direction and magnitude of change) scaled by the learning rate.
  - $W = W - \alpha \cdot dW$
  - $b = b - \alpha \cdot db$
  - Where α is the learning rate and dW, db are the computed gradients from backpropagation.
- **Learning Rate**: Controls how big the steps in updating the parameters are. A smaller learning rate means slower but more precise updates, while a larger rate means faster but possibly more erratic updates.
- **Return Updated Parameters**: The updated W_1, b_1, W_2, and b_2 are returned to be used in the next iteration.

Kmit

1. **No Learning (No Optimization)**:

- If you skip updating the parameters (weights W and biases b) with their gradients, the parameters remain the same throughout the training process.
- As a result, the network doesn't adjust itself to minimize the cost function. The output will remain almost the same, and the model will fail to approximate or classify the data correctly.

Why Updating Parameters is Crucial:

In machine learning, especially in neural networks:

- **Weights and biases** represent the learned knowledge of the model.
- **Backpropagation** computes how much each parameter should change based on the current prediction error.
- **Gradient descent** (or other optimization techniques) uses this information to update the parameters in a way that minimizes the error.

Without updating parameters, the model would never "learn" anything from the data and remain static, which defeats the purpose of training a model.

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