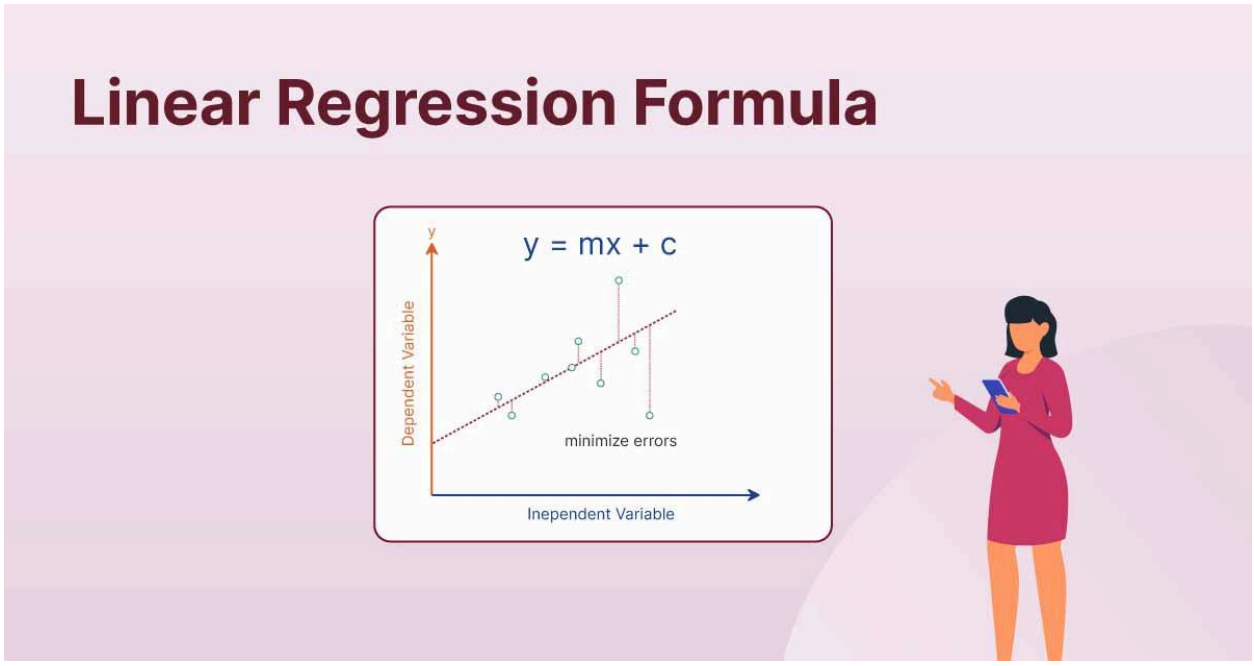
Summer of Code:- Introduction to Deep learning(Id-96)

Week 1: Introduction to Neurons, Basic ML, Logistic & Linear Regression

In the first week, we began by understanding the fundamental concepts of Machine Learning, including supervised and unsupervised learning. We also explored the structure of an artificial neuron—its inputs, weights, bias, and activation function.

We learned how **linear regression** is used for predicting continuous values and how **logistic regression** is used for binary classification tasks. Key mathematical formulations and loss functions (like Mean Squared Error for linear regression and Binary Cross-Entropy for logistic regression) were covered.

Linear regression:-



Week 2: Neural Networks, Gradient Descent, Backpropagation (Assignment 1)

(Assignment 1)

This week focused on building and training simple neural networks. We studied the gradient descent algorithm, which adjusts weights to minimize the loss function. We also implemented backpropagation, the process of computing gradients through chain rule of calculus.

We completed Assignment 1, which involved:

- Implementing forward and backward propagation
- Training a basic neural network for binary classification

🔹 1. Neural Network

A **Neural Network** is a computational model inspired by the human brain. It consists of **layers of nodes (neurons)**:

- **Input Layer**: Takes in the input features.
- **Hidden Layers**: Perform computations (linear + activation functions).
- **Output Layer**: Gives the prediction (e.g., class label, numeric value).

Each neuron in a layer is connected to every neuron in the next layer with **weights**, and each connection has a **bias** term.

Mathematically:
$z = w \cdot x + b$
$a = \text{activation}(z)$ where w is weight, x is input, b is bias, a is the output after applying an activation function.

🔹 2. Gradient Descent

Gradient Descent is an optimization algorithm used to **minimize the loss function** by updating the weights and biases.

Steps:

- Compute the **loss** (error between predicted and actual value).
- Calculate the **gradient** (slope of the loss with respect to weights).
- Update weights in the direction that **reduces the loss**.

Update rule:

$W_{new} = w - \eta \cdot \partial L / \partial w$

where:

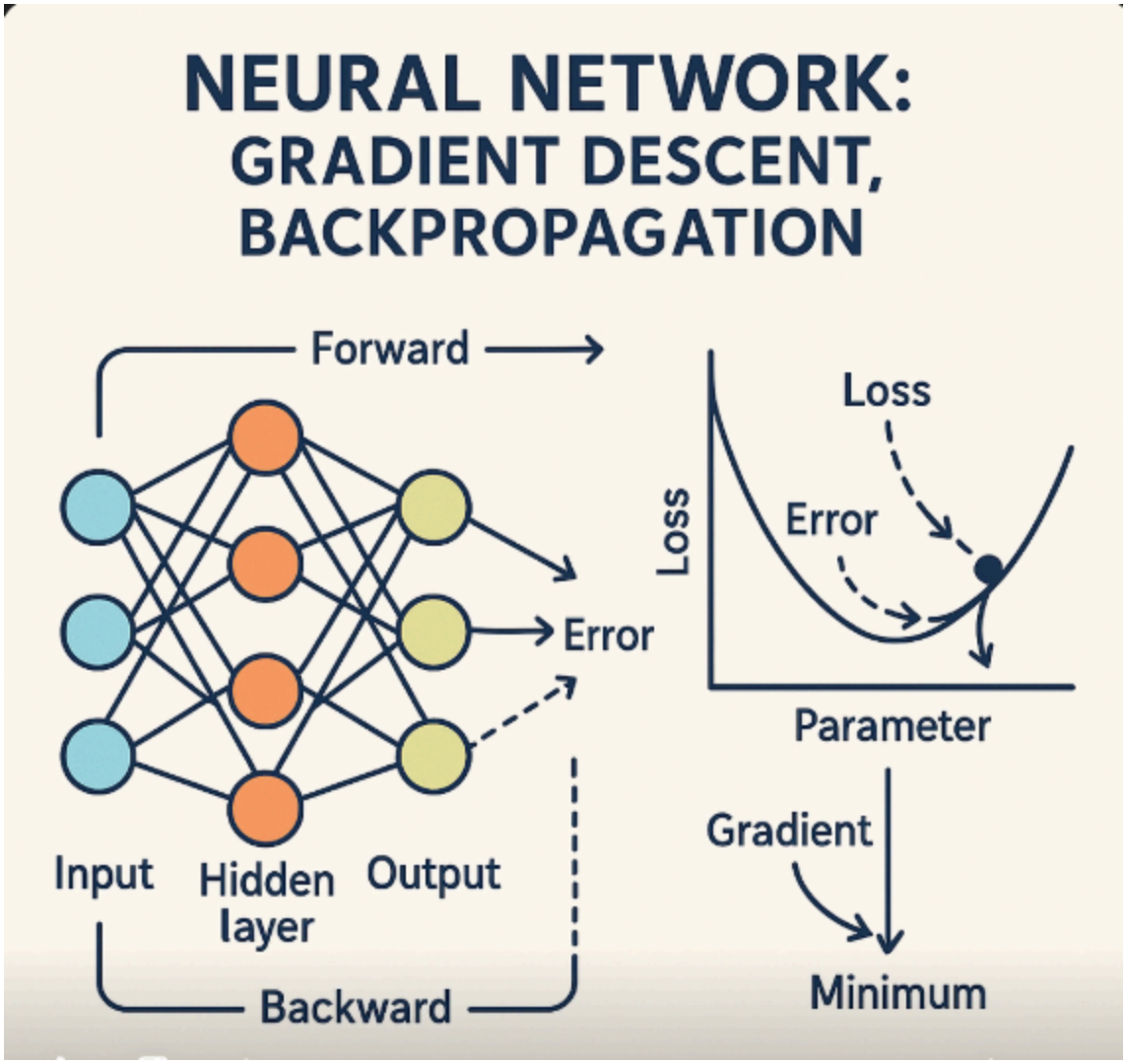
- η is the learning rate,
- $\partial L / \partial w$ is the derivative of the loss w.r.t. weight.

🔹 3. Backpropagation

Backpropagation is the algorithm used to compute the gradients needed for gradient descent.

Steps:

1. **Forward Pass**: Input goes through the network → output is generated.
2. **Compute Loss**: Compare output with the true label.
3. **Backward Pass**: Calculate gradients of the loss w.r.t. weights using the **chain rule** of calculus.
4. **Update Weights**: Using gradient descen



Week 3: Deep Neural Networks (Assignment 2)

In this week, we extended our knowledge to **deep neural networks (DNNs)**, which have more than one hidden layer. We learned about the challenges of deep architectures, such as **vanishing gradients** and **overfitting**, and how techniques like **ReLU activation**, **dropout**, and **batch normalization** help mitigate them.

Assignment 2 required training a multi-layer neural network on a classification dataset, tuning hyperparameters, and evaluating accuracy.

Week 4: Multi-Class Classification (Mini-Project: Digit Classification)

This week introduced **multi-class classification**, where the target variable has more than two classes. We applied **softmax activation** in the output layer and **cross-entropy loss** for training. One-hot encoding of labels and model evaluation using accuracy and confusion matrix were also covered.

We started a **Mini-Project**:
Digit Classification using a CNN on the MNIST dataset

- Built a CNN model using PyTorch
- Used convolution, pooling, and fully connected layers
- Achieved ~98% test accuracy