1. Gradient Descent Purpose:

An **optimization algorithm** used to minimize the **loss function** in models like linear regression, logistic regression, and neural networks.

Working Mechanism:

- Compute the **gradient (derivative)** of the loss function with respect to model parameters.
- Update parameters in the direction that **minimizes the loss**.

Formula:

If θ is a parameter and η is the learning rate:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \cdot \nabla_\theta J(\theta)$$

Where:

- $J(\theta)$ is the loss function (e.g., MSE, cross-entropy).
- $\nabla_\theta J(\theta)$ is the gradient.

We have a dataset with one feature x and one output y. Our goal is to find the best fit line:

$$\hat{y} = \theta_0 + \theta_1 x$$

We'll use **gradient descent** to minimize the **Mean Squared Error**:

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2$$

GRADIENT DESCENT RULE:

For each parameter:

$$\theta_0 := \theta_0 - \eta \cdot \frac{\partial J}{\partial \theta_0}$$

$$\theta_1 := \theta_1 - \eta \cdot \frac{\partial J}{\partial \theta_1}$$

Where the gradients are:

$$\frac{\partial J}{\partial \theta_0} = \frac{2}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)$$

$$\frac{\partial J}{\partial \theta_1} = \frac{2}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) x^{(i)}$$

Let's use this simple dataset:

x	y
1	2
2	3
3	4

Initial values:

- $\theta_0 = 0$
- $\theta_1 = 0$
- Learning rate $\eta = 0.1$

Step 1: Predictions

$$\hat{y}_i = \theta_0 + \theta_1 x_i = 0$$

Residuals: $\hat{y} - y = [-2, -3, -4]$

Step 2: Gradients

$$\frac{\partial J}{\partial \theta_0} = \frac{2}{3}(-2 - 3 - 4) = \frac{-18}{3} = -6$$

$$\frac{\partial J}{\partial \theta_1} = \frac{2}{3}(-2 \cdot 1 - 3 \cdot 2 - 4 \cdot 3) = \frac{2}{3}(-2 - 6 - 12) = \frac{-40}{3} \approx -13.33$$

Step 3: Update Parameters

$$\theta_0 := 0 - 0.1 \cdot (-6) = 0.6$$

$$\theta_1 := 0 - 0.1 \cdot (-13.33) = 1.333$$

Result after 1 iteration:

- $\theta_0 = 0.6$
- $\theta_1 = 1.333$

These values are closer to the best fit line $y = x + 1$.

2. Gradient Boosting (GB) Purpose:

Gradient Boosting is a popular ensemble learning algorithm that combines multiple weak models to create a strong predictive model. It uses Gradient Descent as an optimization algorithm to update the model's parameters.

Working Mechanism:

Initialize the model with a simple weak learner (e.g., a decision tree).
Compute the loss function for the current model.
Calculate the negative gradient of the loss function (pseudo-residuals).
Train a new weak learner to predict the pseudo-residuals.
Update the model by adding the new weak learner with a learning rate (α).
Repeat steps 2-5 for a specified number of iterations.

Formula:

Let $F_m(x)$ be the model after m iterations:

$$F_m(x) = F_{m-1}(x) + \eta \cdot h_m(x)$$

Where:

- $h_m(x)$ is the m-th weak learner (tree) trained on residuals.
- η is the **learning rate**.
- Residuals: $r_i^{(m)} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]$

3. XGBoost (Extreme Gradient Boosting) Purpose:

XGBoost is an optimized implementation of Gradient Boosting that provides faster computation and improved performance. It uses a gradient-based optimization algorithm and introduces additional features like regularization and tree pruning.

Formula:

XGBoost minimizes a **regularized objective function**:

$$\mathcal{L}(t) = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{k=1}^{t} \Omega(f_k)$$

Where:

- $l(y, \hat{y})$ is the loss function (e.g., MSE, log-loss),
- $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$
- T = number of leaves, w_j = weight of leaf j,
- γ, λ are regularization parameters.

Working Mechanism:

- Initialize the model with a simple weak learner (e.g., a decision tree).
- Compute the loss function for the current model.
- Calculate the negative gradient of the loss function (pseudo-residuals).

- Train a new weak learner to predict the pseudo-residuals using a Taylor series expansion of the loss function.
- Update the model by adding the new weak learner with a learning rate (α).
- Regularize the model using L1 and L2 regularization terms.
- Repeat steps 2-6 for a specified number of iterations.

Example

Dataset:

Feature	Target
1	2
2	4
3	6
4	8
5	10

Goal: Predict the target variable using Gradient Boosting with 2 iterations.

Step 1: Initialize the model

- Initialize the model with a simple weak learner (e.g., a decision tree with 1 node).
- Let's assume the initial prediction is the mean of the target variable: **F0(x) = 6 (mean of 2, 4, 6, 8, 10)**

Step 2: Calculate the loss function and pseudo-residuals (Error which is y – y-hat)

- Use Mean Squared Error (MSE) as the loss function: L = (1/2) * (y - F0(x))^2

Calculate the pseudo-residuals (negative gradient of the loss function): r_i = -dL/dF0(x_i) = y_i - F0(x_i)

Feature	Target	Pseudo-residual
1	2	-4
2	4	-2
3	6	0
4	8	2
5	10	4

Step 3: Train a new weak learner

Train a new decision tree to predict the pseudo-residuals.

Decision Tree Prediction:

- If x ≤ 2: predict -3 (avg of -4, -2)
- If 2 < x ≤ 5:
  - If x = 3: predict 0
  - If x > 3: predict 1 (for x = 4) and 3 (for x = 5)

Predicted values:

Let's assume the new decision tree predicts:

Feature	Prediction
1	**-3**
2	**-1**
3	**0**
4	**1**
5	**3**

Step 4: Update the model

- Update the model using a learning rate (α): F1(x) = F0(x) + α * h(x)
- Let's assume α = 0.5: F1(x) = 6 + 0.5 * h(x)

Feature	F0(x)	h(x)	F1(x)
1	**6**	**-3**	**6 + 0.5 * -3 = 4.5**
2	**6**	**-1**	**6 + 0.5 * -1 = 5.5**
3	**6**	**0**	**6 + 0.5 * 0 = 6**
4	**6**	**1**	**6 + 0.5 * 1 = 6.5**
5	**6**	**3**	**6 + 0.5 * 3 = 7.5**

Step 5: Repeat steps 2-4 for the second iteration

- Calculate the new pseudo-residuals: r_i = -dL/dF1(x_i) = y_i - F1(x_i)

Feature	Target	F1(x)	Pseudo-residual
1	**2**	**4.5**	**-2.5**
2	**4**	**5.5**	**-1.5**
3	**6**	**6**	**0**
4	**8**	**6.5**	**1.5**
5	**10**	**7.5**	**2.5**

- Train a new decision tree to predict the pseudo-residuals.
- Let's assume the new decision tree predicts:

Feature	Prediction
1	**-2**
2	**-1**
3	**0**
4	**1**
5	**2**

- Update the model: $F2(x) = F1(x) + \alpha * h(x)$
- Let's assume $\alpha = 0.5$: $F2(x) = F1(x) + 0.5 * h(x)$

Feature	F1(x)	h(x)	F2(x)
1	**4.5**	**-2**	**4.5 + 0.5 * -2 = 3.5**
2	**5.5**	**-1**	**5.5 + 0.5 * -1 = 5**
3	**6**	**0**	**6 + 0.5 * 0 = 6**
4	**6.5**	**1**	**6.5 + 0.5 * 1 = 7**
5	**7.5**	**2**	**7.5 + 0.5 * 2 = 8**

The final predicted values after 2 iterations are: [3.5, 5, 6, 7, 8].