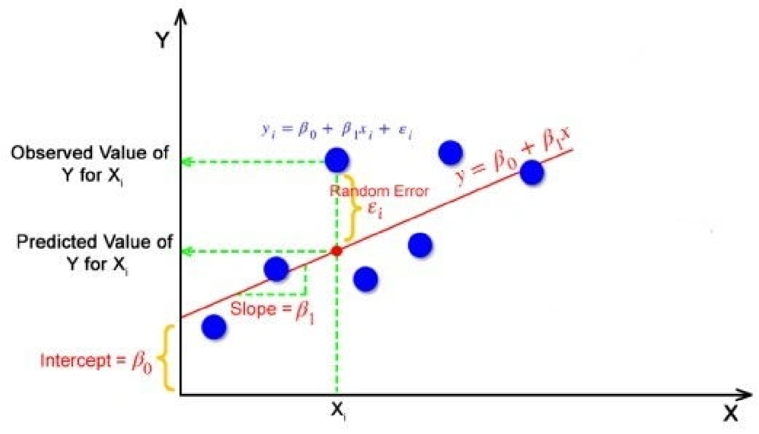
What is Linear Regression?

Linear regression is a statistical method used to model the relationship between a dependent variable (target) and one or more independent variables (features). It is one of the simplest and most widely used regression techniques in machine learning.

- **Simple Linear Regression** involves one independent variable.
- **Multiple Linear Regression** involves multiple independent variables.

Linear regression **assumes a linear relationship** between the dependent and independent variables. The goal is to find the line (or hyperplane in multiple dimensions) that best fits the data.


Y
$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$
Observed Value of Y for X_i
Random Error ε_i
$y = \beta_0 + \beta_1 x$
Predicted Value of Y for X_i
Slope = β_1
Intercept = β_0
X_i
X

1. Standard Equation: $Y = mX + c$

- **Components:**
    - Y: Dependent variable (target/predicted value).
    - X: Independent variable (feature/input).
    - m: Slope of the line (rate of change of Y with respect to X).
    - c: Intercept (value of Y when $X = 0$).

- **Interpretation:** This is the most straightforward representation for a single-variable linear regression (simple linear regression). The slope m indicates how much Y changes for a unit increase in X.

2. General Equation for Multiple Variables: $Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_nX_n$

- **Components:**
  - Y: Dependent variable.
  - $X_1, X_2, \ldots, X_n$: Independent variables.
  - b_0: Intercept term.
  - $b_1, b_2, \ldots, b_n$: Coefficients for each independent variable.
- **Interpretation:** Each coefficient b_i represents the change in Y for a one-unit change in X_i, keeping all other variables constant.

Single-Variable Linear Regression Notation

1. Hypothesis Function

$$h_\theta(X) = \theta_0 + \theta_1 X$$

- **Explanation:**
  - $h_\theta(X)$: Predicted value of Y for a given input X.
  - θ_0: The intercept term, the value of $h_\theta(X)$ when $X = 0$.
  - θ_1: The slope of the line, representing the rate of change of $h_\theta(X)$ with respect to X.
  - X: The input feature (independent variable).

This notation is most commonly used.

Goal of Linear Regression

The primary goal of linear regression is to find a **best-fit line** (linear relationship) that minimizes the error between the predicted values and the true values.

Finding the Best-Fit Line

To achieve the best-fit line:

- The algorithm optimizes the values of θ_1 and θ_2.
- This optimization ensures the error (difference between predicted Y and actual Y) is minimized.

How Does Linear Regression Minimize Error?

1. **Prediction:**
   - For a given X, the model predicts Y using $h_\theta(X)$.
2. **Error Calculation:**
   - The error for each prediction is the difference between the predicted value $\hat{Y}$ (or $h_\theta(X)$) and the true value Y:

$$\text{Error} = \hat{Y} - Y$$

3. **Objective:**
   - Minimize the sum of squared errors across all data points:

$$\text{Objective: } \min \frac{1}{n} \sum_{i=1}^{n} \left(h_\theta(X^{(i)}) - Y^{(i)} \right)^2$$

4. **Update Parameters:**
   - The model updates θ_1 and θ_2 iteratively to minimize the error until the best-fit line is obtained.

Cost Function

The **cost function or loss function** is a mathematical tool used to measure the error between the predicted values and the actual values. In linear regression, the goal of the cost function is to quantify how well the linear model predicts the target variable.

Why Do We Need a Cost Function?

1. **Evaluate Model Performance:**
   The cost function helps us determine whether our current choice of parameters (θ1 and θ2) provides accurate predictions.
2. **Optimization Goal:**
   The cost function provides a value that needs to be minimized to achieve the best-fit line.

Mean Squared Error (MSE)

The most commonly used cost function in linear regression is the **Mean Squared Error** (MSE). It measures the average of the squared differences between predicted and actual values.

$$J(\theta_1, \theta_2) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - Y^{(i)} \right)^2$$

- **Components:**
  - $J(\theta_1, \theta_2)$: The cost function value for the current parameters θ_1 and θ_2.
  - m: Total number of training examples.
  - $h_\theta(X^{(i)})$: Predicted value for the i-th example, computed as $h_\theta(X^{(i)}) = \theta_1 + \theta_2 X^{(i)}$.
  - $Y^{(i)}$: Actual value for the i-th example.

Why Squared Error?

- Squaring the error ensures it is always positive.
- Emphasizes larger errors more than smaller ones, making the model focus on minimizing significant discrepancies.

Understanding the Components:

1. **Prediction Error:**

   For each training example, the difference between the predicted value and the actual value is:

   $$\text{Error} = h_\theta(X^{(i)}) - Y^{(i)}$$

2. **Squared Error:**

   The error is squared to avoid cancellation of positive and negative errors:

   $$\left(h_\theta(X^{(i)}) - Y^{(i)} \right)^2$$

3. **Mean of Squared Errors:**

   The average squared error across all examples is computed to generalize the model's performance:

   $$\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - Y^{(i)} \right)^2$$

4. **Factor $\frac{1}{2}$:**

   The $\frac{1}{2}$ is included to simplify the derivative calculation during optimization.

Key Insights:

- **Small Cost Value:**
  A smaller $J(\theta_1, \theta_2)$ indicates better model performance (predictions are closer to actual values).
- **Optimization Goal:**
  The objective of linear regression is to find θ_1 and θ_2 that minimize $J(\theta_1, \theta_2)$.

Gradient Descent

Gradient descent is an optimization algorithm used to minimize the cost function $J(\theta_1, \theta_2)$ by iteratively adjusting the parameters θ_1 (intercept) and θ_2 (slope). The updates aim to find the values of θ_1 and θ_2 that minimize the error.

How Does Gradient Descent Work?

1. **Initialize Parameters:**
   Start with initial values for θ_1 and θ_2 (e.g., $\theta_1 = 0$, $\theta_2 = 0$).
2. **Compute Predictions:**
   Use the current values of θ_1 and θ_2 to compute predicted values $h_\theta(X)$.
3. **Calculate Cost Function:**
   Evaluate the cost function $J(\theta_1, \theta_2)$ for the current parameters.
4. **Update Parameters:**
   Adjust θ_1 and θ_2 using the gradient descent formula:
   $$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_1, \theta_2)$$
   where α is the learning rate.
5. **Repeat:**
   Iterate the process until the cost function converges to a minimum.

Example: Predicting Weight from Height

Dataset:

Height (X)	Weight (Y)
150	50
160	55

Step-by-Step Calculation:

1. **Initialize Parameters:**
   Let $\theta_1 = 0$ (intercept) and $\theta_2 = 0$ (slope).
   Learning rate (α) = 0.01.

2. **First Iteration:**

   - **Predictions:**
     Using $h_\theta(X) = \theta_1 + \theta_2 X$, for both data points:

     $$h_\theta(150) = 0 + 0 \cdot 150 = 0, \quad h_\theta(160) = 0 + 0 \cdot 160 = 0$$

   - **Errors:**
     Compute the error for each example:

     $$\text{Error} = h_\theta(X) - Y$$

     For $X = 150, Y = 50$: $0 - 50 = -50$
     For $X = 160, Y = 55$: $0 - 55 = -55$

   - **Gradients:**
     Compute partial derivatives of the cost function with respect to θ_1 and θ_2:

     $$\frac{\partial}{\partial \theta_1} J = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(X^{(i)}) - Y^{(i)} \right)$$

     $$\frac{\partial}{\partial \theta_2} J = \frac{1}{m} \sum^{m} \left(h_\theta(X^{(i)}) - Y^{(i)} \right) X^{(i)}$$

Substituting values:

$$\frac{\partial}{\partial \theta_1} J = \frac{1}{2}\left(-50 + (-55)\right) = -52.5$$

$$\frac{\partial}{\partial \theta_2} J = \frac{1}{2}\left(-50 \cdot 150 + (-55) \cdot 160\right) = -8150$$

- **Update Parameters:**
  Using the gradient descent update rule:

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

$$\theta_2 := \theta_2 - \alpha \cdot \frac{\partial}{\partial \theta_2} J$$

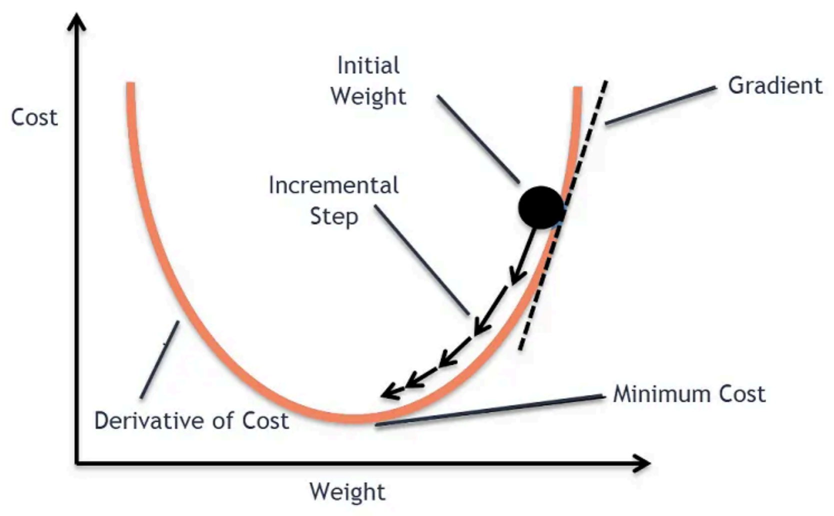
Update θ_1:

$$\theta_1 = 0 - 0.01 \cdot (-52.5) = 0.525$$

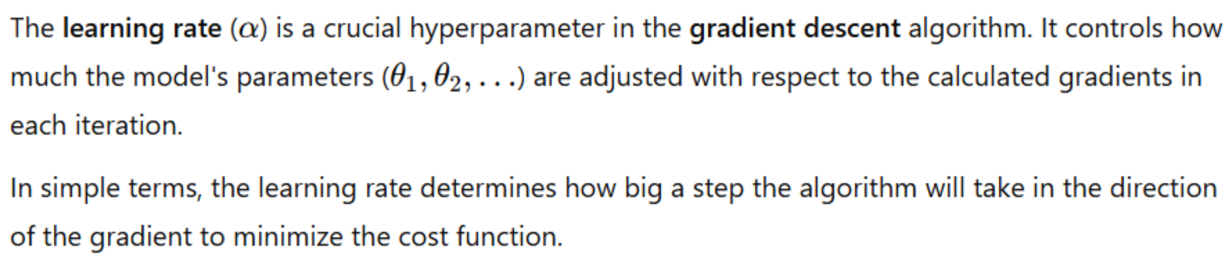
Update θ_2:

$$\theta_2 = 0 - 0.01 \cdot (-8150) = 81.5$$

Second Iteration: Repeat the process with updated $\theta_1 = 0.525$ and $\theta_2 = 81.5$.
Compute new predictions, gradients, and update the parameters iteratively.



Learning Rate in Gradient Descent

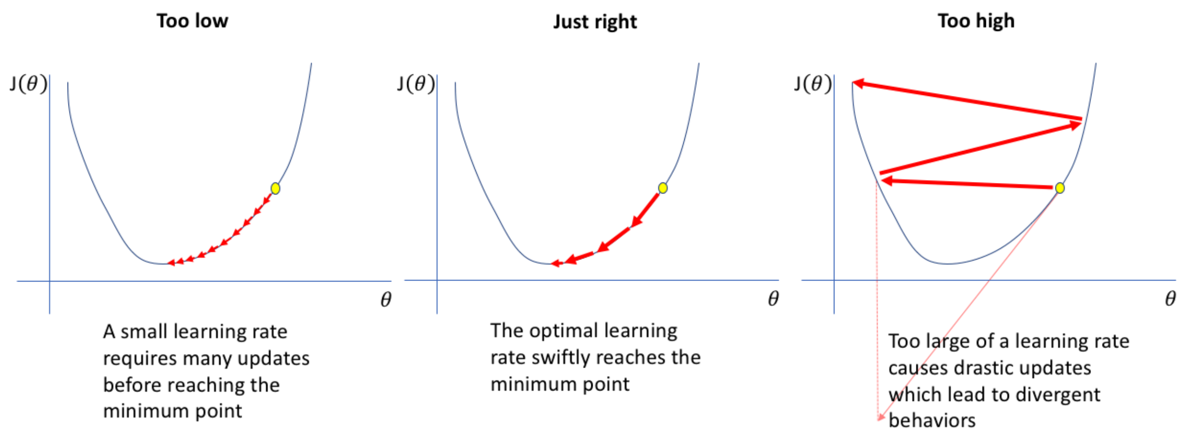

The **learning rate** (α) is a crucial hyperparameter in the **gradient descent** algorithm. It controls how much the model's parameters ($\theta_1, \theta_2, \ldots$) are adjusted with respect to the calculated gradients in each iteration.

In simple terms, the learning rate determines how big a step the algorithm will take in the direction of the gradient to minimize the cost function.

Role of the Learning Rate

The learning rate directly influences the efficiency and success of the gradient descent algorithm:

- **Too Large α:**
  - **Overshooting:** If the learning rate is too large, the updates to the parameters may be too big. This causes the gradient descent algorithm to "overshoot" the minimum and may cause it to diverge, instead of converging to the optimal solution.
  - **Oscillations:** The algorithm may keep jumping back and forth across the minimum, never settling down at a good value.
- **Too Small α:**
  - **Slow Convergence:** If the learning rate is too small, the steps will be too small, causing the gradient descent algorithm to converge very slowly. While it might eventually reach the minimum, it may take an impractically long time to do so.
  - **Risk of Stalling:** If the learning rate is too small, it can get stuck in a local minimum and fail to reach the global minimum.
- **Optimal α:**
  - **Balanced Convergence:** An optimal learning rate strikes a balance. It should be large enough to make meaningful progress towards the minimum, but not so large that it overshoots or oscillates. It allows the algorithm to converge to the optimal solution in a reasonable amount of time.


Too low
Just right
Too high
$J(\theta)$
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Learning Rate Scheduling

In some cases, instead of using a fixed learning rate, the learning rate is adjusted dynamically during training. This can be done in various ways:

- **Decay:**
  Gradually decrease the learning rate as the number of iterations increases. This helps the algorithm take large steps early on and smaller steps as it approaches the minimum.

$$\alpha = \frac{\alpha_0}{1 + \text{decay rate} \times \text{iteration}}$$

Adaptive Methods:

- Algorithms like **AdaGrad**, **RMSprop**, and **Adam** automatically adjust the learning rate during training, making it more flexible and efficient.

Convergence in Optimization Algorithms

Convergence refers to the process in which an optimization algorithm (such as gradient descent) approaches the optimal solution as it iterates. In simpler terms, an algorithm converges when its parameters or variables stop changing significantly, and the solution stabilizes at a point that minimizes the objective (or cost) function.

In the context of machine learning, particularly with **gradient descent** for tasks like **linear regression**, convergence is when the algorithm reaches the best-fit model and the changes in the model's parameters become negligible.

Conditions for Convergence in Gradient Descent

In gradient descent, convergence means that the updates to the parameters are so small that further iterations won't significantly improve the cost function. There are a few conditions that determine when convergence happens:

1. **Small Gradient:**

- The gradient of the cost function with respect to the model's parameters $(\theta_1, \theta_2, \ldots)$ is very small.
- When the gradient approaches zero, the updates to the parameters become negligible, meaning the algorithm has reached a local minimum (or in simple cases, the global minimum).
- This can be mathematically expressed as:
$$\left|\frac{\partial J(\theta)}{\partial \theta_1}\right| < \epsilon \quad \text{and} \quad \left|\frac{\partial J(\theta)}{\partial \theta_2}\right| < \epsilon$$
where ϵ is a very small threshold value.

2. **Small Change in the Cost Function:**

- The change in the cost function between consecutive iterations becomes very small.
- This means that the model's parameters are no longer improving significantly.
- This condition is usually expressed as:
$$\left|J(\theta^{(k)}) - J(\theta^{(k-1)})\right| < \epsilon$$
where $\theta^{(k)}$ is the parameter set at iteration k and $J(\theta)$ is the value of the cost function.

3. **Maximum Number of Iterations:**

- Sometimes, convergence is defined as reaching the maximum number of iterations, even if the cost function or gradient is not exactly zero. This is useful for controlling the runtime of the algorithm.

We need to check this for both theta 0 and 1.

Derivation of Gradient Descent for Linear Regression (with respect to θ_0 and θ_1)

In linear regression, we aim to fit a line to the data by minimizing the **Cost Function** (mean squared error), and this can be achieved by iteratively updating the parameters θ_0 (intercept) and θ_1 (slope) using **Gradient Descent**.

Linear Regression Model:

The hypothesis (prediction) for a linear regression model is:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Where:

- $h_\theta(x)$ is the predicted value for input x,
- θ_0 is the intercept,
- θ_1 is the coefficient (slope) of x.

Cost Function (Mean Squared Error):

The cost function $J(\theta_0, \theta_1)$ measures how well the model fits the data, and it is defined as the **Mean Squared Error (MSE)**:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Where:

- m is the number of training examples,
- $x^{(i)}$ and $y^{(i)}$ are the features and actual values for the i-th training example,
- $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$ is the predicted value for $x^{(i)}$.

The factor $\frac{1}{2m}$ is used to simplify the derivatives when computing the gradients.

Gradient Descent:

The gradient descent algorithm updates the parameters θ_0 and θ_1 iteratively by moving in the direction of the steepest descent (opposite to the gradient of the cost function).

The general update rule for gradient descent is:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

Where:

- α is the **learning rate**,
- $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the partial derivative of the cost function with respect to θ_j.

We will now compute the gradients with respect to θ_0 and θ_1.

1. Gradient with Respect to θ_0:

We need to find the derivative of the cost function $J(\theta_0, \theta_1)$ with respect to θ_0:

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

Substitute $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$:

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

This derivative represents the average of the errors between the predicted values and the actual values.

2. Gradient with Respect to θ_1:

Next, we compute the derivative of the cost function $J(\theta_0, \theta_1)$ with respect to θ_1:

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

Substitute $h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$:

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

This derivative represents the average of the product of the errors and the corresponding input features.

Gradient Descent Update Rules:

Now that we have the gradients, we can update the parameters θ_0 and θ_1 as follows:

1. **Update for θ_0:**

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)} \right)$$

2. **Update for θ_1:**

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x^{(i)}) - y^{(i)} \right) x^{(i)}$$

Iterative Process:

1. **Initialize** θ_0 and θ_1 with random values (e.g., $\theta_0 = 0$ and $\theta_1 = 0$).
2. **Calculate** the gradients $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$ and $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$.
3. **Update** θ_0 and θ_1 using the update rules.
4. **Repeat** the process for several iterations until the parameters converge, i.e., the changes in the parameters become negligible.

Here are the key **assumptions of Linear Regression** that should be met for the model to produce reliable and accurate results:

1. Linearity

- The relationship between the dependent variable y and the independent variable(s) X should be linear.

2. Independence

- The residuals (errors) should be independent of each other.
- This assumption implies that the error for one data point does not provide information about the error for another data point.
- Independence is especially important when data points are temporally or spatially related. If the data points are correlated (e.g., time-series data), then this assumption is violated.

3. Homoscedasticity

- The variance of the error terms should be constant across all levels of the independent variable(s).
- In other words, the spread (or "scatter") of the residuals should be approximately the same for all values of x.
- If the spread of the residuals changes as a function of x, this is known as **heteroscedasticity** and it violates the assumption of homoscedasticity.

4. Normality of Residuals

- The residuals (errors) should be normally distributed.
- This assumption is particularly important when performing statistical tests (such as hypothesis testing or confidence intervals) on the regression coefficients.
- If the residuals are not normally distributed, the statistical significance of the model may be in question.

5. No Multicollinearity (for Multiple Linear Regression)

- In multiple linear regression, the independent variables should not be highly correlated with each other.
- If two or more independent variables are highly correlated, it leads to multicollinearity, which makes it difficult to isolate the individual effects of the independent variables on the dependent variable.
- This can cause the model's coefficients to be unstable and increase the standard errors of the regression coefficients.

6. No Autocorrelation (for Time-Series Data)

- The residuals (errors) should not exhibit autocorrelation, meaning they should not be correlated with each other over time.
- In time-series data, if the residuals are autocorrelated, it indicates that there may be a missing explanatory variable or that the linear regression model does not capture some underlying time-dependent structure.

7. Exogeneity

- The error term ϵ should not be correlated with the independent variables.
- In other words, the predictors X should be exogenous to the error term ϵ. If the predictors are correlated with the error term, this is known as **endogeneity**, which leads to biased and inconsistent estimates of the model parameters.

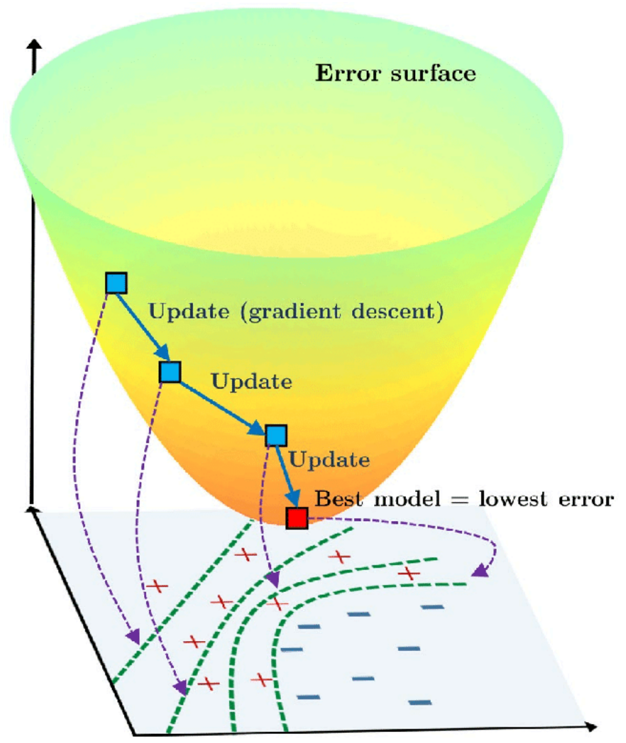
Basics of Graph Shapes in Gradient Descent

1. Univariate (1 Variable) Linear Regression

- **Cost Function**: Mean Squared Error (MSE)
    - **Graph Shape**: The cost function in terms of θ is a **U-shaped parabola**. The goal of gradient descent is to find the minimum point where the cost is lowest.
    - **Process**: Gradient descent adjusts θ iteratively, moving toward the minimum of the parabola by calculating the slope (gradient) and adjusting the parameters.

2. Multivariate (2+ Variables) Linear Regression

- **Cost Function**: Mean Squared Error (MSE)
    - **Graph Shape**: The cost function forms a **bowl-shaped surface** in 3D. Gradient descent updates $\theta 1$ and $\theta 2$ iteratively to reach the lowest point of the bowl.
    - **Contour Plot**: The contours are **elliptical** or **circular**, representing constant cost values.



Performance Metrics: R² and Adjusted R²

When evaluating regression models, two common performance metrics are **R² (coefficient of determination)** and **Adjusted R²**. These metrics help assess how well a model explains the variability in the target variable.

1. R² (Coefficient of Determination)

R² measures the proportion of variance in the dependent variable (Y) that is explained by the independent variables (X) in the model.

Formula:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

where:

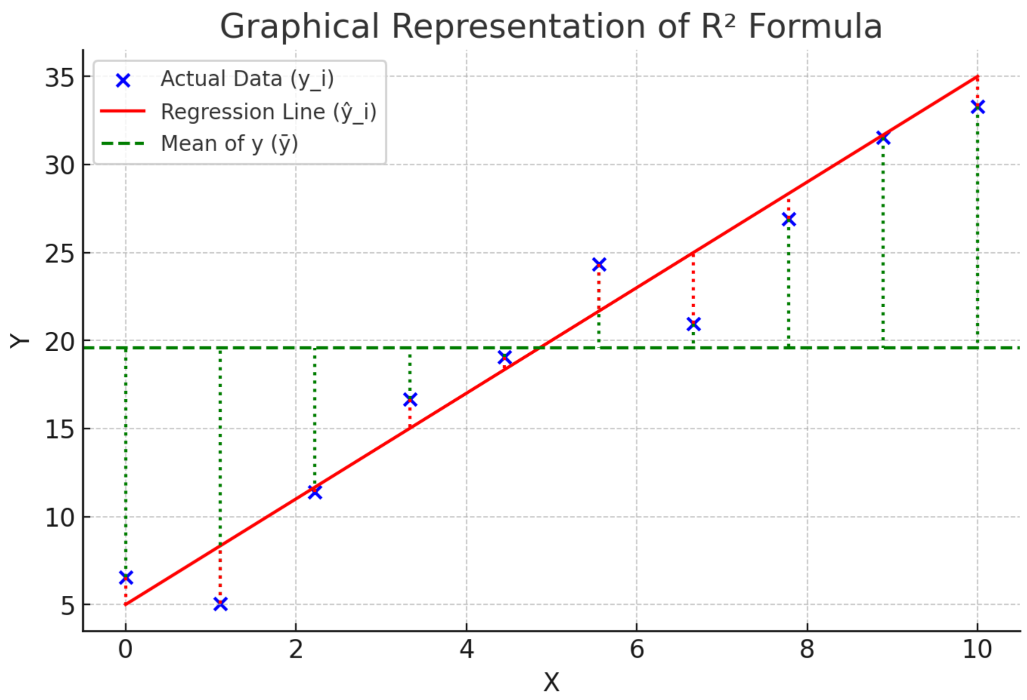
- SS_{res} = **Residual Sum of Squares** $\sum(y_i - \hat{y}_i)^2$ (Error in prediction)
- SS_{tot} = **Total Sum of Squares** $\sum(y_i - \bar{y})^2$ (Total variance in data)
- y_i = Actual value
- $\hat{y}_i$ = Predicted value
- $\bar{y}$ = Mean of actual values

Interpretation:

- R^2 ranges from **0 to 1**:
  - **0**: Model explains none of the variance in Y.
  - **1**: Model perfectly explains all variance.
- Higher R^2 means the model fits the data better.

Limitations of R²:

- Adding more independent variables **always increases** R2, even if they are irrelevant.
- It does not penalize unnecessary complexity, leading to overfitting.



The graph above visually explains the R^2 formula:

- **Blue points**: Actual data points (y_i).
- **Red line**: Regression line (predicted values $\hat{y}_i$).
- **Green dashed line**: Mean of y ($\bar{y}$).
- **Green dotted lines**: Represent **Total Sum of Squares (SS_tot)**, showing how actual values deviate from the mean.
- **Red dotted lines**: Represent **Residual Sum of Squares (SS_res)**, showing errors between actual and predicted values.

2. Adjusted R²

Adjusted R² improves on R² by **penalizing** models that add irrelevant variables. It accounts for the number of predictors and adjusts for model complexity.

Formula:

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - k - 1} \right)$$

where:

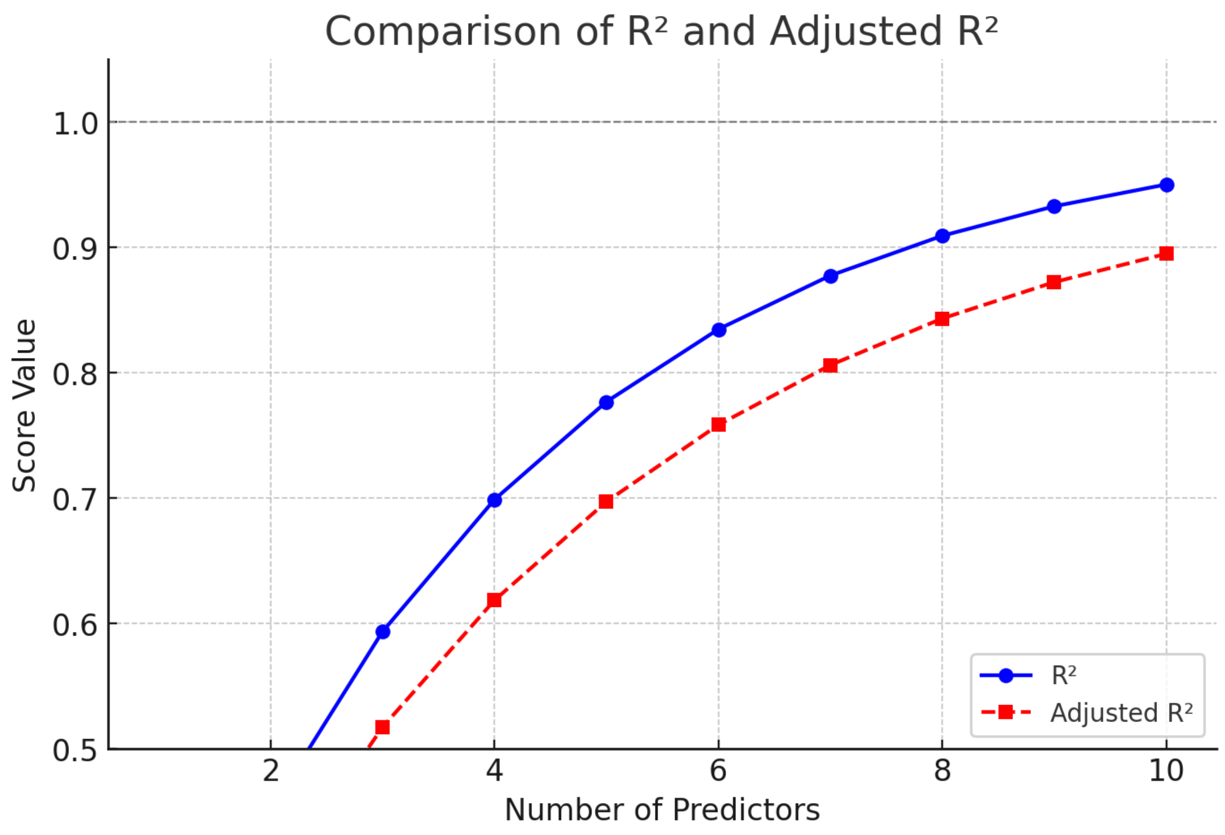
- n = Number of observations (data points)
- k = Number of independent variables (predictors)

Key Differences from R^2:

- If a new variable improves the model, **Adjusted R^2 increases**.
- If a new variable is irrelevant, **Adjusted R^2 decreases**.
- Adjusted R^2 **can be lower than R^2**, but it is more reliable for comparing models with different numbers of predictors.

Graphical Representation

Let me generate a visual explanation comparing R2 and Adjusted R2 with an increasing number of features.



Error Metrics: MSE, MAE, and RMSE

Mean Squared Error (MSE)

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

- **Definition:** MSE is the average of the squared differences between actual and predicted values.
- **Purpose:** It penalizes large errors more than small ones due to squaring.
- **Units:** Squared units of the target variable.

✅ Pros

✔ Differentiable, useful for optimization algorithms (gradient descent).
✔ Penalizes large errors more, making it useful when large errors need to be avoided.

❌ Cons

✖ Sensitive to **outliers** (as errors are squared, large deviations contribute disproportionately).
✖ Harder to interpret because the error is squared, making it in different units from the target variable.

2. Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

- **Definition:** MAE is the average of the absolute differences between actual and predicted values.
- **Purpose:** Measures how far predictions are from actual values without squaring.
- **Units:** Same as the target variable.

✅ Pros

✔ Less sensitive to outliers than MSE.
✔ Easier to interpret since the error is in the same unit as the target variable.

❌ Cons

✖ Less useful for gradient-based optimization because the absolute function is not differentiable at zero.
✖ Treats all errors equally, which may not always be ideal.

3. Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- **Definition:** RMSE is the square root of MSE, making it easier to interpret.
- **Purpose:** Similar to MSE but expressed in the same unit as the target variable.
- **Units:** Same as the target variable.

✅ Pros

✔ Easier to interpret than MSE since it's in the same unit as the target variable.
✔ Penalizes large errors while still being useful in optimization.

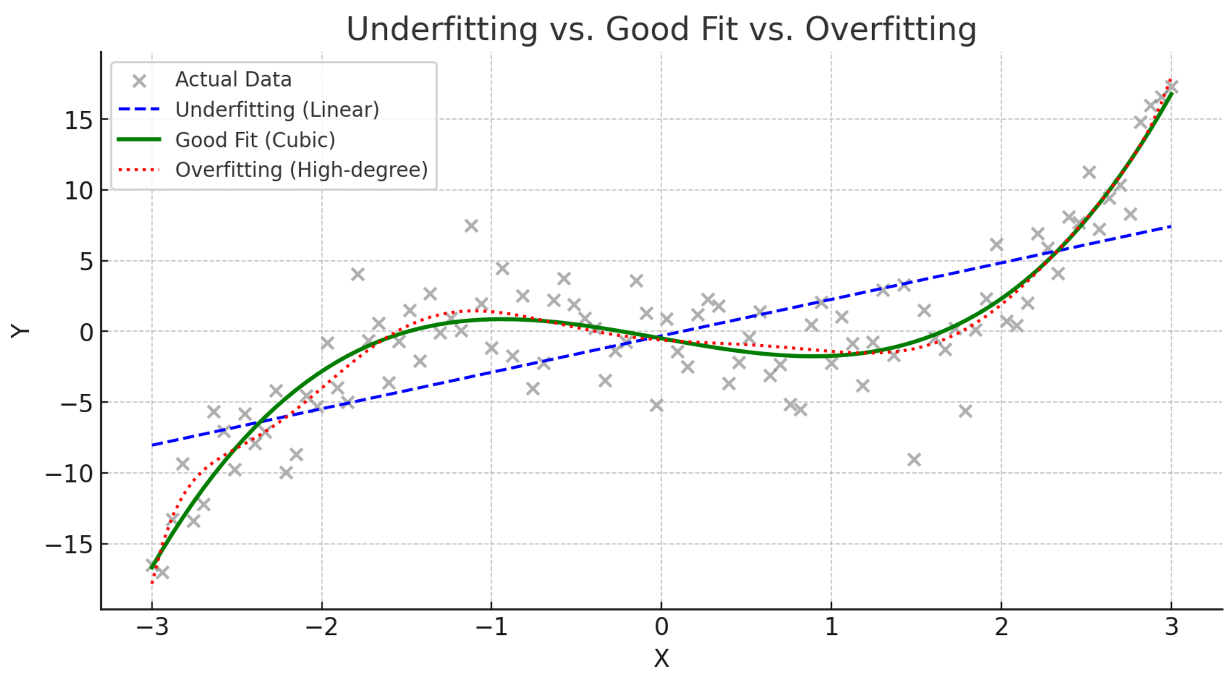
❌ Cons

✖ Still sensitive to outliers (like MSE).
✖ More computationally expensive due to the square root operation.

🔹 **MSE** → When large errors should be penalized more.
🔹 **MAE** → When robustness to outliers is needed.
🔹 **RMSE** → When interpretability in the same unit as the target is important.

Overfitting vs. Underfitting

Concept	Description	Effect on Model	Solution
Underfitting	Model is too simple, failing to capture underlying patterns.	High bias, poor performance on both training & test data.	Use a more complex model, add features, reduce regularization.
Overfitting	Model is too complex, capturing noise along with the pattern.	High variance, excellent performance on training data but poor generalization to new data.	Use regularization (L1/L2), reduce model complexity, increase training data.



Train-Validation-Test Split

Dataset	Purpose
Training Set (60-80%)	Used to train the model.
Validation Set (10-20%)	Used to tune hyperparameters & avoid overfitting.
Test Set (10-20%)	Used to evaluate final model performance.

Validation in Machine Learning

Why is Validation Important?

Validation is used to **fine-tune hyperparameters** and **prevent overfitting**. It helps in selecting the best model before final testing.

Types of Validation Methods

1. Hold-Out Validation

- Splits data into **training (e.g., 80%)** and **validation (e.g., 20%)**.
- Used for **large datasets** when computational efficiency is important.
- **Limitation**: The model's performance depends on how the split is made.

---

2. k-Fold Cross-Validation (CV)

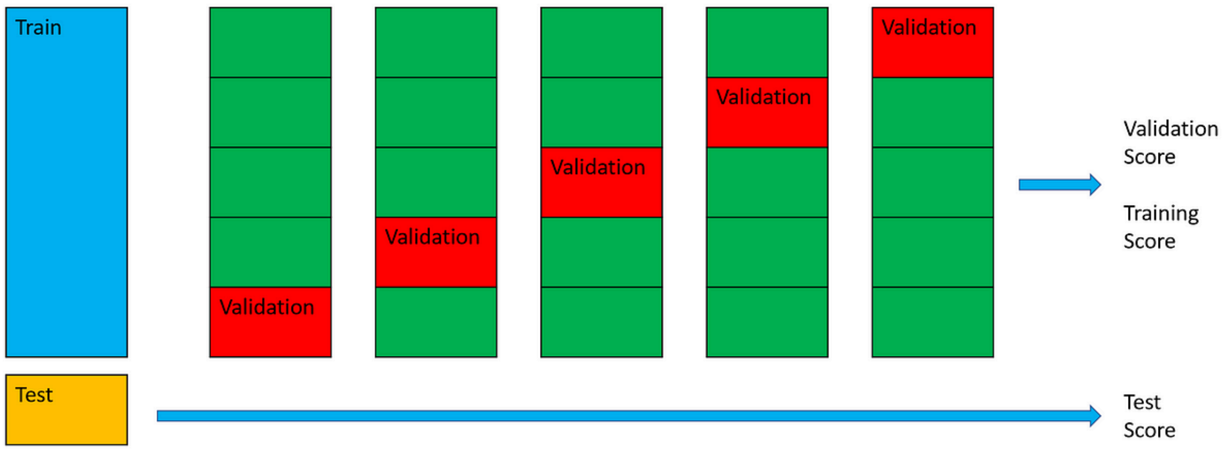
- The dataset is divided into **k equal parts** (e.g., k=5).
- The model trains on **(k-1) folds** and validates on the remaining **1 fold**.
- This process is repeated **k times**, and the final score is the average of all runs.

Advantages

✔ More reliable than hold-out validation.
✔ Reduces variance in model performance estimation.

Common k values

- **k=5 or k=10** is most commonly used.
- **k=2 or k=3** for small datasets to save computation time.



---

3. Stratified k-Fold Cross-Validation

- Used in **classification problems** where class distribution is imbalanced.
- Ensures each fold has a similar proportion of classes as the original dataset.

---

4. Leave-One-Out Cross-Validation (LOOCV)

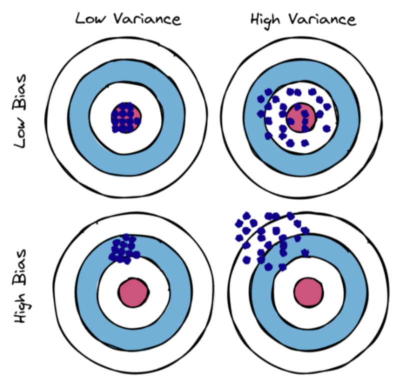
- Each sample is treated as a separate validation set, while the rest are used for training.
- Suitable for **very small datasets**, but computationally expensive.

When to Use Which Validation Technique?

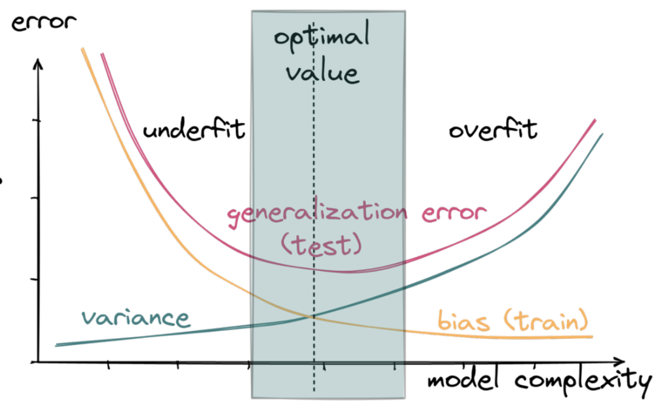
Method	Use When
Hold-Out Validation	Large datasets with clear separation between training and validation.
k-Fold Cross-Validation	Moderate-sized datasets where variance in model performance needs to be minimized.
Stratified k-Fold CV	Classification tasks with imbalanced data.
LOOCV	Very small datasets where every data point matters.

How Validation is Used in Model Training?

1. **Split the dataset** into training and validation sets.
2. **Train the model** using the training set.
3. **Evaluate on the validation set** to tune hyperparameters (e.g., learning rate, number of layers).
4. **Repeat steps 2-3** until the best model is found.
5. **Test the final model** on the test set to check generalization.



pictureA



pictureB

Concept	Definition	Effect on Model
Bias	Error due to **incorrect assumptions** in the learning algorithm.	High bias leads to underfitting. The model is too simple to capture the data patterns.
Variance	Error due to **sensitivity to small fluctuations** in the training data.	High variance leads to overfitting. The model learns noise instead of the true pattern.

Linear Regression using Ordinary Least Squares (OLS)

$$y = \beta_0 + \beta_1 X + \epsilon$$

where:

- y = Dependent variable (target)
- X = Independent variable (feature)
- β_0 = Intercept (value of y when $X = 0$)
- β_1 = Slope (change in y per unit change in X)
- ϵ = Error term (difference between actual and predicted values)

The **OLS method** estimates the parameters (β0 and β1) by minimizing the **sum of squared errors (SSE)**:

$$SSE = \sum (y_i - \hat{y}_i)^2$$

where $\hat{y}_i = \beta_0 + \beta_1 X_i$ is the predicted value.

The **OLS estimates** for β_0 and β_1 are:

$$\beta_1 = \frac{\sum (X_i - \bar{X})(y_i - \bar{y})}{\sum (X_i - \bar{X})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{X}$$

where:

- $\bar{X}$ and $\bar{y}$ are the means of X and y.

We derive the equations for estimating β_0 (intercept) and β_1 (slope) using **Ordinary Least Squares (OLS)** for simple linear regression.

Deriving the Normal Equations

Step 1: Compute Partial Derivatives

$$\frac{\partial}{\partial \beta_0} \sum (y_i - \beta_0 - \beta_1 X_i)^2 = 0$$

$$\frac{\partial}{\partial \beta_1} \sum (y_i - \beta_0 - \beta_1 X_i)^2 = 0$$

Expanding these equations, we obtain:

1. **Summation of errors is zero (Mean of residuals is zero):**

$$\sum (y_i - \beta_0 - \beta_1 X_i) = 0$$

2. **Minimization condition for β_1:**

$$\sum X_i (y_i - \beta_0 - \beta_1 X_i) = 0$$

3. Solving for β_0 and β_1

Step 1: Compute β_1 (Slope)

Rearrange the second equation:

$$\sum X_i y_i - \beta_0 \sum X_i - \beta_1 \sum X_i^2 = 0$$

Substituting $\beta_0 = \bar{y} - \beta_1 \bar{X}$ from the first equation:

$$\beta_1 = \frac{\sum (X_i - \bar{X})(y_i - \bar{y})}{\sum (X_i - \bar{X})^2}$$

Step 2: Compute β_0 (Intercept)

$$\beta_0 = \bar{y} - \beta_1 \bar{X}$$

where:

- $\bar{X} = \frac{1}{n} \sum X_i$ (mean of X)
- $\bar{y} = \frac{1}{n} \sum y_i$ (mean of y)